VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV POČÍTAČOVÝCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF COMPUTER SYSTEMS

# VESTAVĚNÝ SYSTÉM PRO AUTOMATIZACI CHOVNÉ STANICE AKVARIJNÍCH RYB

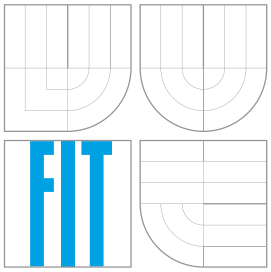
DIPLOMOVÁ PRÁCE
MASTER'S THESIS

AUTOR PRÁCE
AUTHOR

Bc. FILIP MOC

BRNO 2018

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV POČÍTAČOVÝCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF COMPUTER SYSTEMS

# VESTAVĚNÝ SYSTÉM PRO AUTOMATIZACI CHOVNÉ STANICE AKVARIJNÍCH RYB

EMBEDDED SYSTEM FOR AUTOMATION OF AQUARIUM FISH BREEDING STATION

DIPLOMOVÁ PRÁCE
MASTER'S THESIS

AUTOR PRÁCE
AUTHOR

Bc. FILIP MOC

VEDOUCÍ PRÁCE
SUPERVISOR

Ing. VÁCLAV ŠIMEK

BRNO 2018

# Abstrakt

Tato diplomová práce se zabývá tvorbou vestavěného systému pro automatickou bezobslužnou obměnu vody v akváriích. Krom obměny vody vestavěný systém řídí také osvětlení a ventilaci v odchovně akvarijních ryb. V této zprávě naleznete informace o technologiích vhodných pro automatizaci odchovny akvarijních ryb. Dále jsou zde informace o návrhu a implementaci mého konkrétního finálního funkčního řešení.



# Abstract

This master's thesis is concerned about embedded system for automatic water renewal in aquariums. Besides water renewal system also controls lights and ventilation in aquarium fish breeding station. In this report you will find informtions about technology relevant to automation of aquarium fish breeding station. Also you will find here informations about design and implementation of my specific final working solution.



# Klíčová slova

senzory teploty, senzory výšky hladiny, elektrické ventily, čerpadla vody, vestavěné systémy, 1-Wire, SPI, I²C, Bootloader, ATMega32, DS18B20, HC-SR04, ENC28J60.

# Keywords

temperature sensors, water level sensors, electric valves, water pumps, embedded systems, 1-Wire, SPI, I²C, Bootloader, ATMega32, DS18B20, HC-SR04, ENC28J60.



# Citace

Filip Moc: Vestavěný systém pro automatizaci chovné stanice akvarijních ryb. Brno, 2018. Diplomová práce. Vysoké učení technické v Brně, Fakulta informačních technologií. Vedoucí práce Ing. Václav Šimek.

**Vestavěný systém pro automatizaci chovné stanice akvarijních ryb**

# Prohlášení

Prohlašuji, že jsem tuto diplomovou práci vypracoval samostatně pod vedením pana Václava Šimka.

Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

.....
Filip Moc
22. května 2018

# Poděkování

Děkuji mému vedoucímu panu Václavu Šimkovi, který mi poskytl technické rady a podporu při vypracovávání této diplomové práce.

Děkuji svým rodičům za odborné rady z oboru akvaristiky.

# Obsah

# Kapitola 1

# Úvod

Naše rodina provozuje odchovnu akvarijních ryb, ve které chováme akvarijní ryby pro účely komerčního prodeje. V odchovně je umístěno přibližně 250 akvárií o celkovém objemu přibližně $30\,m^3$. Základními předpoklady pro úspěšný chov akvarijních ryb jsou především: udržování kvalitní vody v akváriích (čistota, teplota, prokysličenost, tvrdost a PH vody), správné krmení ryb a zajištění vhodného osvětlení. Pro udržení čisté vody v akváriích, je nutné vodu pravidelně obměňovat.

Záměrem této diplomové práce je vytvořit vestavěný systém pro automatickou bezobslužnou obměnu vody v akváriích. Systém bude nasazen v naší odchovně akvarijních ryb. Zajistit je potřeba zejména, aby bylo obměněno správné množství vody za jednotku času a aby teplota vody vpouštěné do akvárií byla ve stanoveném rozsahu. Systém bude dále řídit osvětlení a větrání. Cílem této automatizace je především nahradit manuální práci, kterou musí obsluha výměně vody a celkově provozu odchovny věnovat, a ušetřit tak obsluze významné množství pracovního času.

V kapitole 2 se budu zabývat obecnými požadavky na komerční chov akvarijních ryb a teoretickými možnostmi, jak provoz odchovny automatizovat.

V kapitole 3 se budu zabývat některými existujícími technologiemi a pojmy, které budu v dalších kapitolách využívat.

V kapitole 4 se budu zabývat některými komponentami, které by mohly být pro automatizaci chovu akvarijních ryb vhodné. Některé z těchto komponent využiji při návrhu výsledného vestavěného systému pro automatizaci řízení odchovny.

V kapitole 5 navrhnu řídící systém odchovny, který budu později implementovat a nasazovat do provozu.

V kapitole 6 připravím zapojení externích periferií k počítačům ATMega32.

V kapitole 7 implementuji bootloader pro jednočipové počítače ATMega32 vybavený možností vzdáleného flashování firmwaru.

V kapitole 8 implementuji aplikační programy pro jednočipové počítače ATMega32, které budou dohromady řídit vestavěný systém pro automatizaci řízení odchovny.

V kapitole 9 implementuji aplikace pro přístupový server, pomocí kterého je možné vestavěný systém vzdáleně ovládat a také zajišťuje logování a monitorování chodu vestavěného systému.

V kapitole 10 implementuji uživatelskou aplikaci pro obsluhu.

V kapitole 11 předvedu některé možné analýzy logovaných dat.

Na závěr v kapitole 12 shrnu výsledky realizace a nasazení vestavěného systému do provozu.

# Kapitola 2

# Požadavky a postupy pro zajištění automatizace řízení chovné stanice akvarijních ryb

Na obrázku 2.1 je fotografie naší odchovny pro ilustraci.

**Obrázek 2.1:** *Fotografie odchovny*

Z literatury lze ohledně chovu akvarijních ryb doporučit [8][4].

## 2.1 Obměna vody

Akvarijním rybám je potřeba pravidelně obměňovat vodu. To znamená, že část původní vody z těchto akvárií je potřeba odpustit a novou vodu doplnit. Toto lze provádět manuálně, avšak manuální výměna vody při větším množství akvárií (konkrétně v naší odchovně je

přibližně 250 akvárií) vyžaduje značné množství času, které musí obsluha činnosti věnovat. Cílem je proto obměnu vody automatizovat - tedy nahradit práci člověka strojem, aby se časové požadavky na obsluhu snížili.

Odpouštění staré vody z akvárií lze realizovat několika způsoby. Například odčerpání čerpadlem nebo vypuštění ventilem. Vybavit každé akvárium elektricky ovládaným ventilem, nebo dokonce čerpadlem, by bylo značně finančně nákladné. Akvária v naší odchovně, jsou vybavena přepadem, který zajistí, že pokud hladina vody stoupne nad určitou mez, začne voda sama odtékat přepadem do odpadu. Přepad takto jednoduše řeší celý problém odpouštění staré vody z akvárií a zabývat se je tedy potřeba pouze napouštěním nové vody.

Před vpuštěním vody do akvárií je potřeba napouštěnou vodu příslušně připravit. Zejména je nutné zajistit její ohřev na správnou teplotu. Ohřev vody by bylo možné realizovat i při průtoku vody rovnou z vodovodu. Ale v naší odchovně jsme pro zjednodušení přípravy vody použili přípravnou nádrž. Objem nádrže je přibližně $800\,l$. Voda bude nejdříve napuštěna do této nádrže, v nádrži bude voda připravena (především tedy ohřáta na požadovanou teplotu). A když bude voda připravena, bude čerpadlem čerpána dál do rozvodu pro akvária.

Díky použití nádrže je snadnější zajistit ohřev vody na přesně požadovanou teplotu. Dále je možné vodu v nádrži dodatečně prokysličovat, případně jinak upravovat. Odstátá voda je navíc pro akvarijní ryby vhodnější. Takové řešení také počítá s možností využívání více zdrojů vody. Konkrétně naše odchovna bude nově moct díky nádrži využívat krom veřejného vodovodu také vodu čerpanou z vrtu. Prozatím nevíme, v jakém poměru budeme moct využít vody z vrtu. Bude záležet na kvalitě vody z vrtu. Z finančních důvodů je žádoucí maximalizovat využití vody z vrtu. V ideálním případě, pokud nenarazíme na žádné problémy s vodou z vrtu, bude pro napouštění využívána výhradně voda z vrtu a veřejný vodovod bude sloužit pouze jako záložní zdroj vody.

Po přípravě je potřeba vodu dopravit do akvárií. Taktéž vybavovat každé akvárium elektricky ovládaným ventilem nebo čerpadlem kvůli zajištění dopouštění vody, by bylo finančně příliš nákladné. Levnějším řešením je k akváriím přivést společný přívod vody s dostatečným tlakem a k jednotlivým akváriím vést pouze hadičky zakončené tryskami. Trysky by měly mít průřez v poměru přibližně odpovídající objemu akvária, aby v každém akváriu bylo dopuštěno dostatečné množství vody a naopak v žádném akváriu nebylo dopuštěno zbytečně příliš velké množství vody. Nadměrná obměna vody by měla navíc negativní dopad na kvalitu vody v akváriu.

Pokud by byl ale zase pro všechna akvária pouze jeden společný přívod vody, vzniklo by několik problémů. Bylo by obtížné zajistit stejný tlak u všech akvárií. Na tlak má vliv například délka přívodního potrubí/hadic, která je pro různá akvária odlišná. Čerpadlo by muselo být schopno dodávat vysoký průtok, aby obsloužilo všechny trysky a stále byl tlak v přívodu vody dostatečný, nebo by naopak průřez trysek musel být velmi malý. U trysek s malým průřezem by byl problém zajistit dostatečnou přesnost velikosti průřezu. Navíc se dá u malých průřezů očekávat zanášení drobnými nečistotami a tudíž nízká spolehlivost. Výsledné řešení by nebylo dobře škálovatelné (zapojení dalších akvárií do systému obměny vody by způsobilo snížení tlaku v přívodu vody u všech stávajících akvárií, které by bylo nutné kompenzovat). Je proto potřeba zvolit vhodný kompromis mezi řešením se samostatným přívodem pro každé akvárium (příliš drahé) a jediným společným přívodem (příliš nepraktické). V odchovně jsou akvária rozmístěna do několika regálů, a tudíž se nabízí jako vhodný kompromis použít samostatný přívod vody pro každý regál.

Jednotlivé přívody vody je potřeba nějakým způsobem řídit, aby bylo možné získat kontrolu nad tím, jaké množství vody do jednotlivých regálů proteče. Jednou z možností by bylo použít více čerpadel. Průtok by pak bylo možné řídit přiváděním napájení na

jednotlivá čerpadla. Podstatně levnější variantou je použít pouze jedno čerpadlo a průtok do jednotlivých regálů ovládat elektrickými ventily. Jediná nevýhoda použití ventilů oproti více čerpadlům je, že není možné dopouštět vodu do více regálů současně (čerpadlo by nedokázalo udržet dostatečný tlak). Z praktického hlediska je však tato nevýhoda nevýznamná.

V naší odchovně budeme používat jedno čerpadlo s automatickým tlakovým spínačem, které bude čerpat vodu z nádrže do rozvodu pro vypouštění na požadovaný tlak. Vypouštění do jednotlivých regálů budeme ovládat elektrickými ventily.

## 2.2 Osvětlení

Krom obměny vody je akvarijním rybám potřeba zajistit také vhodný pravidelný režim osvětlení. Akvarijní ryby nemají příliš individuální potřeby na osvětlení a je tedy možné řízení osvětlení zjednodušit na jednoduché úkony, kdy ráno je potřeba rozsvítit a večer zhasnout. Některé druhy ryb ale mohou na náhlé a prudké rozsvícení reagovat silným úlekem, při kterém hrozí i úhyn vystresované ryby. Mnohem šetrnější je tedy rozsvěcení provádět plynule.

V naší odchovně je krom hlavního osvětlení (zářivky a LED zdroje světla) také tlumené žárovkové osvětlení. Proces rozsvěcení bude tedy rozdělen do tří kroků. Nejdříve se rozsvítí tlumená světla, po uplynutí nějakého času (přibližně 5 minut) se rozsvítí hlavní osvětlení. Následně se za několik vteřin tlumené osvětlení zhasne. Překryv tlumeného a hlavního osvětlení je použit z toho důvodu, že zářivkám nějaký čas trvá, než se rozsvítí a jejich svit se stabilizuje. Proces zhasínání bude probíhat podobně, akorát v opačném pořadí.

Rozsvěcení bude naplánováno na 9:00 a zhasínání na 22:00.

## 2.3 Krmení

Akvarijní ryby je potřeba také krmit. Tedy do akvárií pravidelně vsypávat vhodné množství krmiva vhodného druhu, přičemž vhodné množství a vhodný druh krmiva závisí na druhu, velikosti a počtu akvarijních ryb v akváriu. Příliš malé množství krmiva způsobí, že ryby nebudou mít zajištěnu dostatečnou obživu. Naopak příliš velké množství krmiva se zase negativně projeví na kvalitě vody v akváriu. I v oblasti krmení akvarijních ryb je velký prostor pro automatizaci, nicméně automatizaci krmení prozatím neplánujeme.

## 2.4 Větrání

V odchovně je potřeba zajistit dostatečnou kvalitu vzduchu. To je potřeba k zajištění vhodného prostředí pro obsluhu a dále také k zajištění dostatečného prokysličování vody v akváriích. Pro udržení kvalitního prostředí je potřeba v odchovně zajistit občasné větrání. V odchovně máme momentálně dva ventilátory. Jeden ventilátor zajišťuje přísun čerstvého vzduchu z venku a druhý ventilátor zajišťuje cirkulaci vzduchu mezi místnostmi. Nedostatečné větrání by způsobilo nekvalitní prostředí a naopak příliš časté větrání by v zimním období způsobilo zbytečně velké tepelné ztráty, které by se museli kompenzovat topením. U venkovního ventilátoru je proto důležité vhodně zvolit dobu, po kterou bude ventilátor zapnutý. Odhadovaná vhodná doba je přibližně 8 % času. Vnitřní ventilátor by mohl běžet nepřetržitě, ale dostačující je, aby běžel pouze 25% času.

# Kapitola 3

# Existující technologie a pojmy

V této kapitole uvádím některé existující technologie a technické pojmy, které budu v dalších kapitolách používat.

## 3.1 Ethernet

Ethernet [3] je v současné době nejpoužívanější protokol linkové vrstvy v oblasti počítačových sítí. Komunikace protokolem ethernet probíhá v paketech. Každý paket obsahuje MAC adresu cíle, MAC adresu zdroje, EtherType nebo délku rámce, přenášená data (payload) a kontrolní součet (CRC).

## 3.2 1-Wire

1-Wire [5] je komunikační protokol pro sběrnici vyvinutý firmou Dallas Semiconductor. 1-Wire pro komunikaci v obou směrech používá jeden komunikační vodič. Do sběrnice 1-Wire je možné připojit několik zařízení, přičemž každé zařízení má svou jedinečnou identifikaci. Krom samotné komunikace je možné komunikační vodič použít kombinovaně zároveň jako napájení. Zařízení komunikující po sběrnici 1-Wire jsou vybavena parazitní kapacitou, která zajišťuje napájení po čas, kdy je komunikační vodič aktivní (v log. 0).

## 3.3 SPI

SPI (Serial Peripheral Interface) je synchronní sériové komunikační rozhraní umožňující oboustrannou komunikaci. V rámci rozhraní je jednou stranou master, který ovládá hodinový signál. Protistranou je slave, který je hodinovým signálem řízen. Krom hodinového signálu jsou použity další dva datové signály MOSI (Master Output Slave Input) a MISO (Master Input Slave Output). Při každém hodinovém cyklu je přenesen jeden bit přes signál MOSI a jeden bit přes signál MISO.

Při použití rozhraní SPI se obvykle také počítá se signálem Slave Select, pomocí kterého je možné zvolit slave zařízení, je-li slave zařízení připojeno více. Slave Select se používá také pro synchronizaci (reset rozhraní) a s některými zařízeními může být použití signálu Slave Select vyžadováno i v případě, že je připojeno pouze jedno slave zařízení.

## 3.4 I²C

I²C (Inter-Integrated Circuit) je sériová sběrnice podporující připojení více master a více slave zařízení. I²C používá dva vodiče SDA (Serial Data Line) a SCL (Serial Clock Line). Slave zařízení je na sběrnici adresováno pomocí adresy o délce 7 bitů. Na sběrnici je tedy možné připojit nejvýše 128 slave zařízení.

Komunikace na sběrnici je zahájena stavem start condition, kdy je hodnota signálu SDA změněna z log. 1 na log. 0, zatímco se signál SCL nachází v log. 1 a komunikace je ukončena stavem stop condition, kdy je hodnota signálu SDA změněna z log. 0 na log. 1, zatímco se signál SCL nachází v log. 1. Během komunikace je dovoleno signál SDA měnit pouze v čase, kdy se hodnota signálu SCL nachází v log. 0.

Hodnotu signálu SCL řídí zejména master, nicméně i slave může hodnotu signálu SCL pozdržet v log. 0 a tím komunikaci pozastavit. Master v takové situaci čeká, dokud slave signál SCL neuvolní do log. 1.

## 3.5 Bootloader

Bootloader (zavaděč) v oblasti vestavěných systémů je program, který je spuštěn před předáním řízení hlavnímu aplikačnímu programu. Za běžných okolností je hlavním účelem bootloaderu spustit aplikační program. V této práci využiji bootloader zejména k umožnění vzdálené aktualizace aplikačního programu přes Ethernet.

# Kapitola 4

# Výběr komponent pro realizaci systému

Tak jako u každého běžného vestavěného systému, tak i u tohoto, který je předmětem mé diplomové práce, je předpokladem, že se skládá z řídícího počítače a vstupně/výstupních komponent. V této kapitole se věnuji těm komponentám, které považuji za vhodné pro použití v systému pro automatickou obměnu vody. Zejména jde o senzory, pomocí kterých je možné reagovat na výšku hladiny, senzory teploty, a to především takové, kterými je možné měřit teplotu vody a také možnosti, jak tok vody ovládat - tedy například elektricky ovládané ventily nebo elektrická čerpadla vody. Nevynechám ani obvyklé univerzální komponenty, jako jsou jednočipové počítače.

## 4.1 Teplotní senzory

Teplotní senzor je potřeba především pro měření teploty vody v nádrži. Podle teploty vody v nádrži se pozná, zda je potřeba vodu ohřívat, či už má voda správnou teplotu. Výhledově bychom také chtěli měřit teplotu vzduchu (venkovní i vnitřní) a zohlednit tyto teploty při řízení větrání.

### 4.1.1 DS18B20 [5]

- Digitální teplotní senzor
- Rozhraní 1-Wire (viz kapitola 3.2)
- Pracuje na napětích 3.0 V až 5.5 V
- Teplotní rozsah −55 °C až 125 °C
- Rozlišení na 0.0625 °C
- Běžně dostupný i ve vodotěsném pouzdře s kabelem

### 4.1.2 SMT172 [10]

- Teplotní senzor s PWM výstupem
- Pracuje na napětích 2.7 V až 5.5 V

- Teplotní rozsah −45 °C až 130 °C
- Rozlišení na 0.5 °C (0.1 °C - pouzdro TO18)

### 4.1.3 LM75A [9]

- Digitální teplotní senzor
- Rozhraní I²C (viz kapitola 3.4)
- Pracuje na napětích 2.8 V až 5.5 V
- Teplotní rozsah −55 °C až 125 °C
- Rozlišení na 0.125 °C

## 4.2 Senzory výšky hladiny

Senzor výšky hladiny je potřebný pro měření výšky hladiny v nádrži. Na základě tohoto měření se pozná, zda je potřeba nádrž dopouštět, či zda už je nádrž dostatečně napuštěna.

### 4.2.1 Plovákové spínače

- Na trhu je mnoho druhů.
- Mají nižší hustotu než voda (plavou na hladině).
- Fungují na principu mechanického spínače, který se sepne/rozepne podle toho, jestli se plovákový spínač nachází pod vodou, nebo na hladině.
- Spínač stačí uchytit (například za provázek nebo přívodní vodiče) pod hladinou vody.
  - Délkou provázku/vodiče je možné zvolit v jaké výšce hladiny bude spínač spínat.
  - Pokud provázek/vodič dosáhne svou délkou k hladině, spínač se sepne/rozepne.

### 4.2.2 Mechanické tlakové hladinové spínače

- Fungují na principu mechanického vzduchového tlakového spínače.
  - Voda stlačuje vzduch v hadičce - čím vyšší hladina, tím vyšší tlak vzduchu.
- Typicky mají více mechanických přepínacích kontaktů s hysterezí a nastavitelnou citlivostí.
  - Stav/poloha kontaktů má vliv na objem vnitřní nádoby a tedy i na tlak - kontakty se vzájemně ovlivňují.
- Pokud při používání nedochází k uvolnění tlaku (hadička je stále pod hladinou), stlačený vzduch se časem vytrácí a nastavená citlivost se rozlaďuje.

### 4.2.3 HC-SR04 [2]

- Ultrazvukový senzor vzdálenosti
- Napájení 5 V
- Jednoduché TTL rozhraní
  - vstup pro ovládání ultrazvukového vysílače
  - výstup formou impulzu, jehož délka odpovídá době návratu ultrazvukových vln
- Funguje na principu aktivního sonaru.
  - Senzor při měření vyšle ultrazvukový signál, který se odrazí od překážky zpět do senzoru.
  - Z časové prodlevy mezi vysláním signálu a přijetím odraženého signálu je možné určit vzdálenost.
- Měří ve vzdálenostech 2 $cm$ až 4 $m$.
- Ultrazvukový signál se dobře odráží od hladiny vody, čehož je možné využít k měření výšky hladiny.

## 4.3 Elektrické ventily

Obecně je elektrický ventil zařízení, které se po připojení elektrického napětí otevře, nebo uzavře, a tím umožní, nebo zabrání průtoku tekutiny. Typicky jsou elektrické ventily v klidovém stavu uzavřeny a při přivedení elektrického napětí se otevírají. Existují však i varianty ventilů, které jsou v klidovém stavu otevřeny a při přivedení napětí se uzavírají.

Elektrické ventily jsou potřeba pro ovládání napouštění nádrže a také vypouštění vody z nádrže do jednotlivých regálů s akvárii.

### 4.3.1 Elektromagnetické ventily ovládané přímo

- Jádro elektromagnetu přímo otevírá hlavní zábranu, která brání průtoku tekutiny.
- Při odpojení napětí se zábrana vrací do klidové polohy a průtok vody uzavře.
- Ventil může správně fungovat i bez tlakového rozdílu.
- Při příliš velkém tlakovém rozdílu může být ventil neschopen zábranu otevřít.

### 4.3.2 Elektromagnetické ventily ovládané nepřímo

- Jádro elektromagnetu otevírá pouze vedlejší kanál.
- Vyrovnání tlakového rozdílu způsobí otevření hlavního kanálu.
- Pro správnou funkci ventilu je zapotřebí, aby tlakový rozdíl překročil minimální hranici stanovenou pro daný ventil, jinak nemusí dojít k úplnému otevření ventilu.

## 4.4 Elektrická čerpadla

Obecně je elektrické čerpadlo zařízení, které při přivedení elektrického napětí začne dodávat kinetickou energii v daném směru přivedené tekutině. Jinými slovy začne čerpat tekutinu (např. vodu) z jednoho místa na jiné místo. Čerpadel existuje mnoho druhů (např. pístová, lamelová, zubová). Čerpadla mohou být vybavena tlakovým spínačem, který zajistí, že dosáhne-li tlak na výstupu čerpadla určité hranice, spínač se rozepne a čerpadlo se zastaví. Dále mohou být čerpadla vybavena zpětným ventilem nebo zpětnou klapkou, což zabraňuje průtoku tekutiny v opačném směru.

Jedno čerpadlo je potřeba pro čerpání vody z nádrže do rozvodu pro vypouštění do akvárií. Další čerpadlo je potřeba pro čerpání vody z vrtu.

## 4.5 Jednočipové počítače

Jednočipové počítače jsou potřeba pro řízení jednotlivých komponent vestavěného systému. Jednočipové počítače musí zejména pracovat se senzory a zpracovávat naměřené hodnoty a ovládat výstupní komponenty (relé).

### 4.5.1 ATMega32 [1]

- 32 KiB FLASH paměť pro program
- 2 KiB SRAM
- 1 KiB EEPROM pro perzistentní data
- 32 vstupů/výstupů
- Možnost použití bootloaderu a self programování

Bootloader může být libovolný program umístěný ke konci flash paměti ATMega32, který za běžných okolností spustí aplikační program umístěný na začátku flash paměti. Může ale také modifikovat aplikační program a provádět libovolné jiné úkony, které může provádět aplikační program. V případě potřeby může aplikační program předávat řízení zpět bootloaderu a naopak.

### 4.5.2 JY-MEGA32 (Minimum AVR System Board)

- Vývojová deska s ATMega32
- 5 tlačítek
- LED pro některé výstupy
- Pinheader pro 32 vstupů/výstupů
- ISP Konektor pro programování
- JTAG konektor
- USB konektor
- Vestavěný regulátor napětí pro 3.3 V
- Krystal pro oscilátor

## 4.6 Ethernet adaptéry

O ethernetu viz kapitola 3.1. Ethernetové adaptéry jsou potřeba pro zjištění komunikace mezi jednočipovými počítači a vzdáleného ovládání jednočipových počítačů pomocí běžného univerzálního počítače.

### 4.6.1 ENC28J60 [6]

- Podpora 10BASE-T
- Podpora Half-duplex / Full-duplex
- SPI rozhraní (viz kapitola 3.3)
- Pracuje na napětích 3.1 V až 3.6 V
- Na vstupech tolerantní vůči napětí 5 V
- 8 KiB Buffer
- Velké množství chyb [7]
- Dostupné ve formě připravené osazené desky včetně konektoru RJ-45

### 4.6.2 W5100 [11]

- Podpora 10BASE-T a 100-BASE-TX
- Podpora Half-duplex / Full-duplex
- SPI rozhraní (viz kapitola 3.3)
- Pracuje na napětí 3.3 V
- Na vstupech tolerantní vůči napětí 5 V
- 16 KiB Buffer
- Hardwarová podpora pro TCP, UDP, ICMP, IPv4, ARP, IGMP, PPPoE

# Kapitola 5

# Návrh řídícího systému

## 5.1 Jednočipové počítače a komunikace

Pro realizaci systému řízení obměny vody použiji dva jednočipové počítače ATMega32 na vývojových deskách JY-MEGA32 (počítač č. 1 a počítač č. 2). K různým počítačům budou připojeny různé komponenty (viz kapitola 5.2). Počítače spolu budou komunikovat pomocí ethernetu. Pro komunikaci po ethernetu použiji adaptéry ENC28J60. Krom těchto dvou počítačů bude do ethernetové sítě připojen také přístupový server, pomocí kterého bude možné systém ovládat. Přístupový server bude také počítačům umožňovat synchronizaci času a také bude možné pomocí přístupového serveru v případě potřeby aktualizovat firmware v jednočipových počítačích.

Abych zajistil možnost vzdálené aktualizace firmwaru (pomocí ethernetu), implementuji ovladač pro ENC28J60 do bootloaderu ATMega32. Bootloader bude umožňovat jednoduchou komunikaci na rozhraní ethernet, pomocí které bude možno vyčíst aktuální stav flash paměti, operační paměti, modifikovat flash paměť a provést reboot systému. Bootloader bude využívat watchdog periferii. Od aplikačního programu bude vyžadováno, aby občas (nejméně jednou za 2s) předal řízení bootloaderu, aby bootloader mohl zpracovat příchozí ethernet pakety a vyresetovat watchdog. Samotný aplikační program watchdog nikdy resetovat nebude.

Díky tomuto přístupu by neměla chyba vedoucí k zacyklení aplikačního programu vést k zaseknutí celého jednočipového počítače. Pokud nebude chyba v samotném bootloaderu, nemělo by dojít ke ztrátě vzdálené kontroly nad systémem. I kdyby byl do flash paměti nahrán chybný aplikační program, mělo by být možné pomocí bootloaderu nahrát nový aplikační program a funkci systému obnovit.

Zatímco aplikační program bude v obou jednočipových počítačích odlišný, protože počítače budou zajišťovat jiné úkoly, bootloader může být v obou počítačích shodný (jedinou výjimkou bude MAC adresa pro adaptér ENC28J60).

Komunikace nebude na úrovni jednočipových počítačů nijak zabezpečena. Předpokládá se, že zabezpečená bude samotná připojená ethernetová síť (např. oddělená zabezpečená VLAN s přístupem přes VPN). Pro jednoduchost implementace síťové komunikace nebude použit žádný známý protokol síťové vrstvy (např. IPv6 nebo IPv4), ale komunikační protokol bude postaven přímo nad linkovou vrstvou (tedy ethernetem).

## 5.2 Ostatní komponenty / periferie

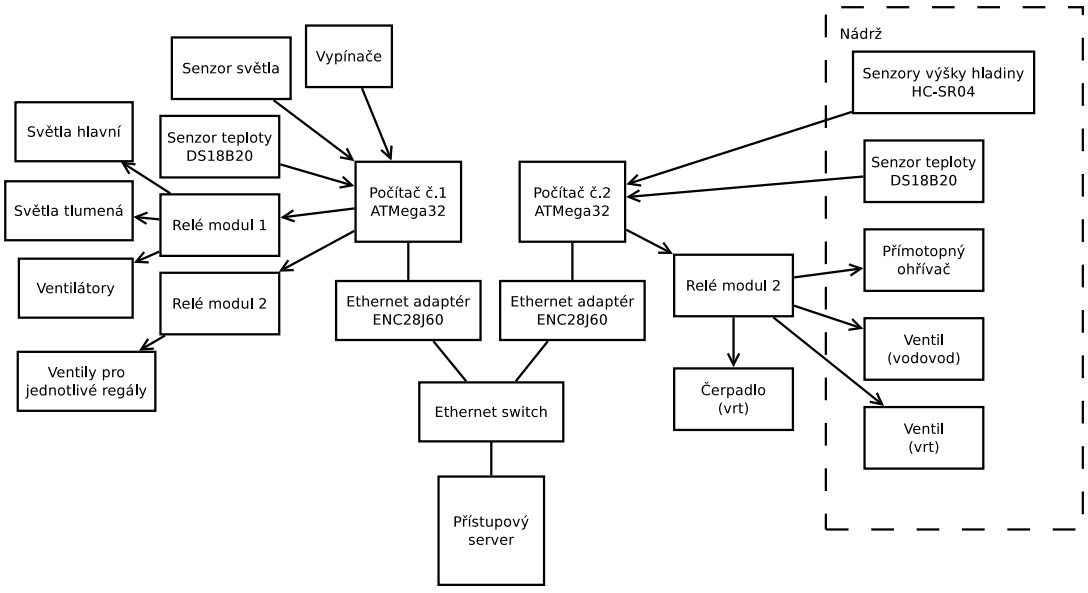
Počítač č. 1 bude ovládat ventily pro vpouštění vody do jednotlivých regálů, osvětlení a větrání. Mezi nádrží a rozvodem pro akvária je umístěno čerpadlo, které v rozvodu zajišťuje tlak. Čerpadlo je spínáno automaticky při poklesu tlaku v rozvodu. Řízení tohoto čerpadla počítačem tedy není nutné. Při zavření ventilů se čerpadlo samo zastaví. Počítač č. 2 bude ovládat napouštění a ohřev vody v nádrži včetně venkovního čerpadla pro čerpání vody z vrtu. Důvodem právě takovéhoto rozdělení je především vzdálenost počítačů k jednotlivým komponentám.

Pro spínání napětí ~230V budou použita relátka (modul s 8 relátky). Pro spínání větších proudových odběrů (čerpadlo, topení) nebudou relátka spínat přímo proudově náročná zařízení, ale bude použit stykač.

Pro měření teploty vody v nádrži bude použit teplotní senzor DS18B20 ve vodotěsném pouzdře připojený k počítači č. 2. Pro měření výšky hladiny bude použita dvojice senzorů HC-SR04 připojená k počítači č. 2. Dvojice je použita pro zajištění vyšší odolnosti proti poruše senzoru a chybnému měření. Aplikační program v počítači č. 2 bude počítat s průměrem měřených hodnot od obou senzorů a bude kontrolovat, zda odchylka mezi měřenými hodnotami není příliš vysoká, což by indikovalo chybné měření.

## 5.3 Schéma navrženého systému

Na obrázku 5.1 je znázorněno navržené schéma systému. Šipky udávají směr přenosu významné informace. Pro jednoduchost nejsou v obrázku vyznačeny stykače.



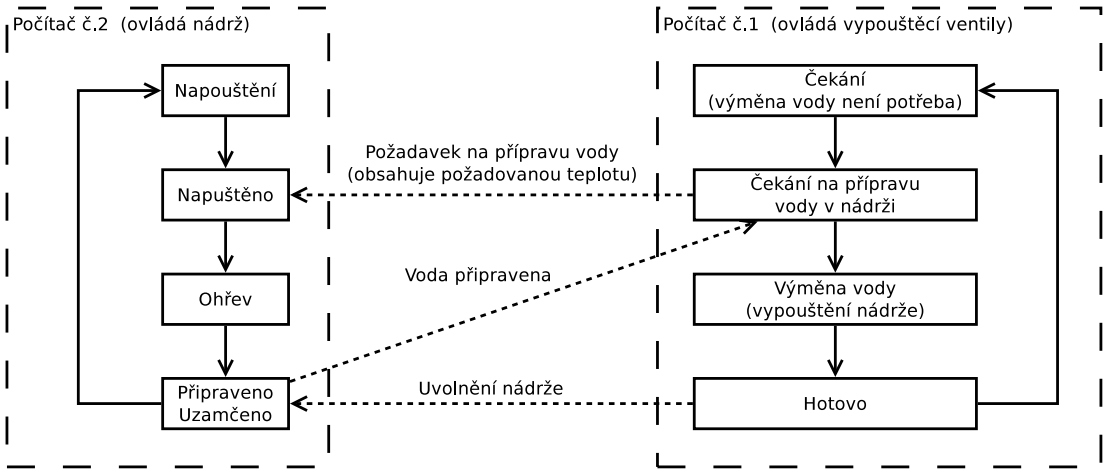

**Obrázek 5.1:** *Schéma systému*

## 5.4 Proces obměny vody

Na obrázku 5.2 je znázorněn navržený proces obměny vody včetně komunikace. Tečkované šipky znázorňují komunikaci. Plné šipky znázorňují přechody mezi jednotlivými stavy. Až po přijetí zprávy (tečkovaná šipka) je dovolen přechod do dalšího stavu. Požadavek na přípravu

vody je počítačem č. 1 zasílán opakovaně, dokud není přijata odpověď od počítače č. 2, že je voda připravena. Po dobu uzamčení nádrže může počítač č. 1 vodu z nádrže vypouštět, zatímco počítač č. 2 nesmí s vodou v nádrži nijak manipulovat, ani ohřívat ani dopouštět, aby nedošlo ke změně teploty vody v nádrži.

Pokud počítač č. 1 během doby, kdy je nádrž uzamčena, nezašle znovu požadavek na uzamčení nádrže, nádrž se po vypršení časového limitu uvolní automaticky a počítač č. 2 přejde do stavu napouštění i bez obdržení zprávy na uvolnění nádrže. Díky tomu počítač č. 2 neuvázne ve stavu uzamčené nádrže v případě, že se zpráva o možném uvolnění nádrže ztratí.




**Obrázek 5.2:** *Diagram obměny vody*

V rámci napouštění bude systém umožňovat míchání vody z obou vodních zdrojů pomocí dvou ventilů (vodovod a vrt). Případně bude nádrž napouštěna výhradně vodou z vrtu a vodovod bude sloužit pouze jako záložní zdroj vody.

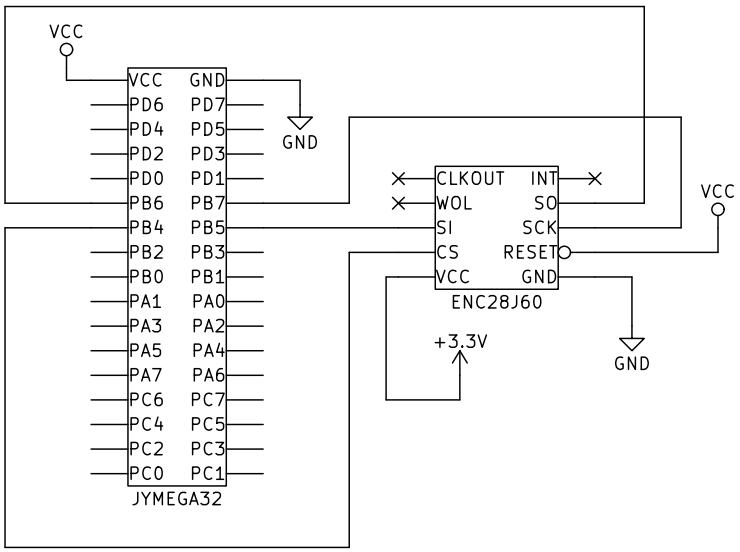
Vypouštění bude prováděno pomocí vypouštěcích ventilů. Každý regál bude mít vlastní vypouštěcí ventil. Experimentálně jsme zjistili, že je každý ventil potřeba otevřít denně celkem na přibližně 20 minut, aby bylo obměněno správné množství vody. To znamená, že ventily by měly být otevřeny přibližně 1.39 % času. Nějaký čas (přibližně $2s$) ale trvá, než se přívod dostatečně natlakuje a průtok vody se stabilizuje. Je proto potřeba neměnit stavy ventilů příliš často, aby se tato doba natlakování významně neprojevila (pokud bude ventil otevřen, měl by být otevřen alespoň na minutu).

# Kapitola 6

# Zapojení periferií

## 6.1 ENC28J60

Na obrázku 6.1 je zobrazeno propojení ethernetového adaptéru ENC28J60 s deskou JY-MEGA32.



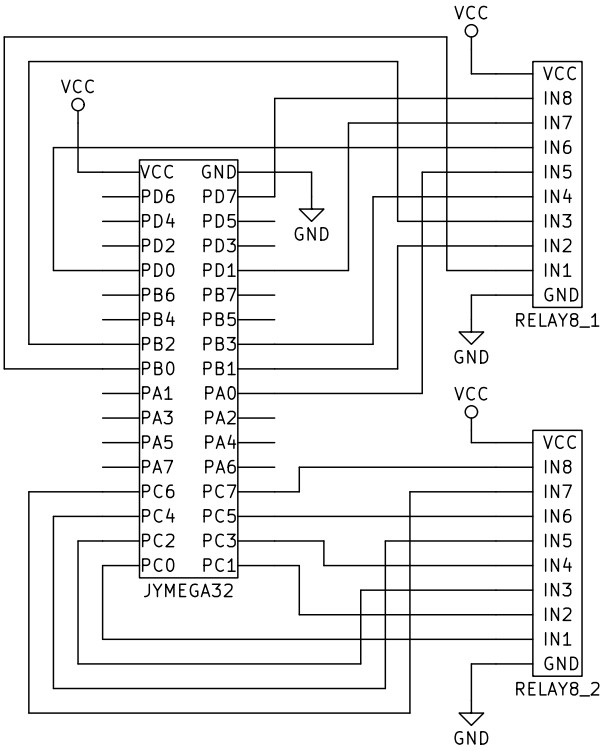

**Obrázek 6.1:** *Schéma propojení JY-MEGA32 a ethernetového adaptéru ENC28J60*

Použití pinů PB5, PB6 a PB7 jako MOSI, MISO a SCK je důležité, protože vnitřní SPI periferie jednočipového počítače ATMega32 je připojena právě na tyto tři piny. Pro Chip-Select by bylo možné použít i jiný pin než PB4, nicméně použil jsem PB4, aby byly piny k ethernetovému adaptéru ENC28J60 logicky u sebe.

Napájení ENC28J60 (pin Vcc) je připojeno k výstupu 3.3V regulátoru na desce JY-MEGA32.

## 6.2 Relé moduly

Na obrázku 6.2 je zobrazeno propojení relé modulů s deskou JY-MEGA32.




**Obrázek 6.2:** *Schéma propojení JY-MEGA32 a relé modulů*

K jednočipovému počítači č.2 je připojen pouze jeden modul (RELAY8_2).
V tabulce 6.1 je znázorněno zapojení jednotlivých výstupních periferií k relé modulům.

| **Počítač č.1 - Relé modul 1** | |
|---|---|
| **Relé** | **Periferie** |
| 1 | Hlavní světla |
| 2 | Tlumená světla |
| 3 | Vnitřní ventilátor |
| 4 | Vnější ventilátor |
| 5 | - |
| 6 | - |
| 7 | - |
| 8 | - |

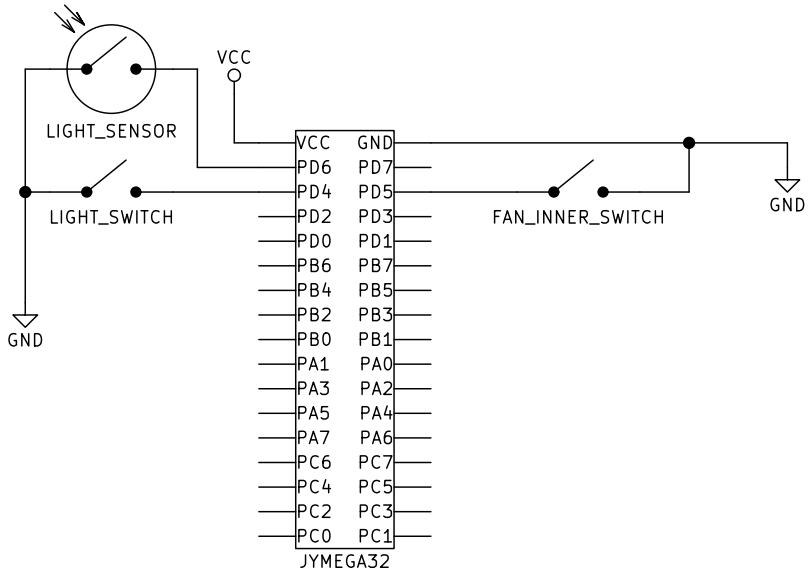
| **Počítač č.1 - Modul 2** | |
|---|---|
| **Relé** | **Periferie** |
| 1 | Ventil 1 |
| 2 | Ventil 2 |
| 3 | Ventil 3 |
| 4 | - |
| 5 | - |
| 6 | - |
| 7 | - |
| 8 | - |

| **Počítač č.2 - Modul 2** | |
|---|---|
| **Relé** | **Periferie** |
| 1 | Přímotopný ohřívač |
| 2 | Ventil - vodovod |
| 3 | Ventil - vrt |
| 4 | Čerpadlo - vrt |
| 5 | - |
| 6 | - |
| 7 | - |
| 8 | - |

**Tabulka 6.1:** *Zapojení výstupních 230V periferií k relé modulům*

## 6.3 Vypínače a senzor světla

Na obrázku 6.3 je zobrazeno připojení vypínačů a senzoru světla k desce JY-MEGA32 jednočipového počítače č.1.

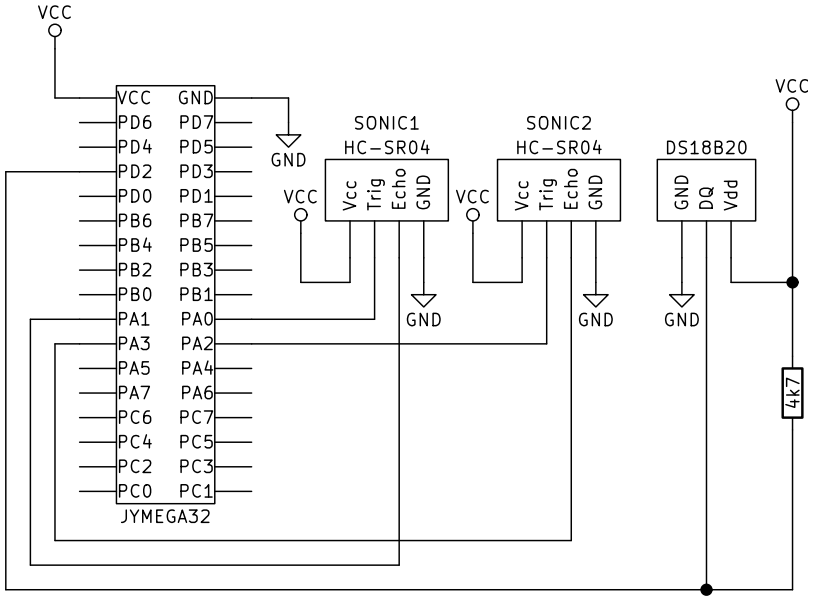



**Obrázek 6.3:** *Schéma propojení JY-MEGA32 (počítač č.1) s vypínači a senzorem světla*

Při rozepnutém vypínači (resp. výstupních kontaktů senzoru světla) je na jednočipovém počítači čtena hodnota log. 1 díky internímu pull-up. V případě sepnutého vypínače (kontaktů senzoru světla) je čtena hodnota log. 0.

## 6.4 Senzory teploty a výšky hladiny

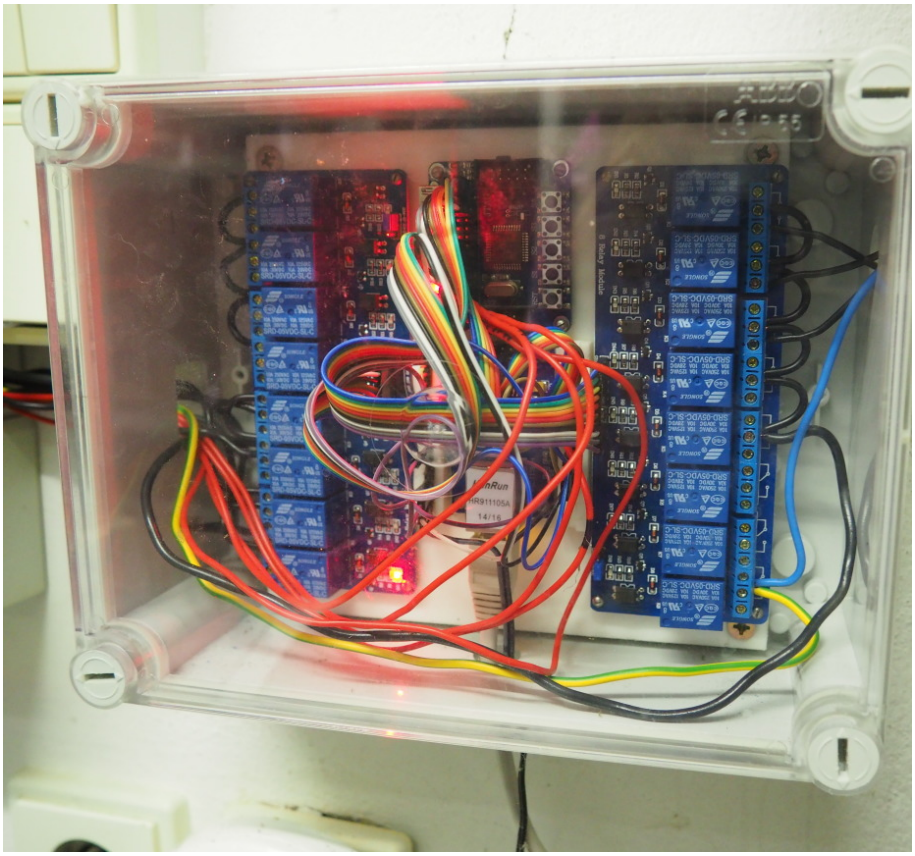
Na obrázku 6.4 je zobrazeno připojení senzoru teploty a senzorů výšky hladiny k desce JY-MEGA32 jednočipového počítače č.2.




**Obrázek 6.4:** *Schéma propojení JY-MEGA32 (počítač č.2) se senzory DS18B20 a HC-SR04*

## 6.5 Fotografie jednočipového počítače č.1

Na obrázku 6.5 je fotografie sestaveného jednočipového počítače č.1 s periferiemi zapojeného do systému.



**Obrázek 6.5:** *Fotografie jednočipového počítače č.1*

Na obou stranách jsou vidět relé moduly. Vlevo je relé modul 1 - ovládání osvětlení a ventilace. Vpravo je relé modul 2 - ovládání vypouštěcích ventilů. Nahoře je umístěn jednočipový počítač č.1 na desce JY-MEGA32. Uprostřed pod ním je modul s ethernetovým adaptérem ENC28J60.

# Kapitola 7

# Implementace bootloaderu pro ATMega32

## 7.1 Fuse a Lock bity

Na počítačích ATMega32 jsem provedl úpravy fuse a lock bitů zejména související s použitím bootloaderu. Následuje konfigurace a vysvětlení některých relevantních fuse a lock bitů.

- `BLB01 = 1, BLB02 = 1` - Aplikační program je odemčen pro čtení i zápis (ponechána výchozí konfigurace).
- `BLB11 = 0, BLB12 = 1` - Bootloader je odemčen pro čtení, ale uzamčen pro zápis.
  - Výchozí konfigurace je `BLB11 = 1` a `BLB12 = 1`. Díky nastavení lock bitu `BLB11 = 0` je bootloader chráněn proti modifikaci. I kdyby byl do počítače nahrán chybný aplikační program, nemělo by se mu podařit smazat bootloader. Nemělo by tedy dojít ke ztrátě vzdáleného přístupu k počítači. Bootloader je stále možné změnit pomocí SPI programování.
- `BOOTRST = 0` - Výchozí konfigurace je `BOOTRST = 1`. Díky nastavení fuse `BOOTRST = 0` je po resetu počítače spuštěn bootloader namísto aplikačního programu.
- `BOOTSZ0 = 0, BOOSTSZ1 = 0` - Aplikační program končí na adrese `0x37FF` (v bajtech `0x6FFF`) a bootloader začíná na adrese `0x3800` (v bajtech `0x7000`). Pro aplikační program je tedy vyhrazeno 28 KiB a pro bootloader je to 4 KiB. To je maximální velikost bootloaderu, kterou lze na ATMega32 nakonfigurovat.

Více o těchto fuse a lock bitech lze nalézt v datasheetu ATMega32 [1] na stranách 256 a 257.

## 7.2 Funkce, které bootloader zajišťuje

- Inicializace watchdogu
- Inicializace ethernetového adaptéru ENC28J60
- Umožnění vzdáleného ovládání včetně flashování
- Zavedení aplikačního programu

- Zpřístupnění ethernetového adaptéru pro aplikační program
- Zpřístupnění funkce na kontrolu paketů pro aplikační program
- Resetování watchdogu

## 7.3 Ovladač ethernetového adaptéru ENC28J60

Pro účely komunikace s ethernetovým adaptérem ENC28J60 jsem implementoval ovladač (soubory `enc28j60.c` a `enc28j60.h`). Tento ovladač využívá vnitřní SPI periferii počítače ATMega32. Pro zapojení viz kapitola 6.1. ENC28J60 pracuje v režimu slave, takže počítač ATMega32 pracuje v režimu master. Ovladač nastavuje adaptér ENC28J60 do režimu full-duplex. Oblast paměti adaptéru `0x0000` až `0x19FF` je použita pro přijímání a oblast paměti `0x1A00` až `0x1FFF` je použita pro odesílání. Díky této konfiguraci je možné odesílat pakety o maximální velikosti, kterou dovoluje ethernet standard [3]. Nastavení přijímacího bufferu na začátek paměti (`0x0000`) je důležité kvůli chybě v adaptéru (viz chyba č.5 v errata dokumentu [7]).

## 7.4 Ethernet komunikace a formát paketů

Tabulka 7.1 znázorňuje formát ethernetových paketů, se kterými bootloader pracuje.

| 6 bajtů | 6 bajtů | 2 bajty | 40 až 1498 bajtů | 2 bajty | 4 bajty |
|---|---|---|---|---|---|
| cílová ethernetová adresa | zdrojová ethernetová adresa | `0x0A01` (EtherType) | Uživatelská data | Aplikační kontrolní součet (CRC-16-CCITT) | Ethernetový kontrolní součet (CRC-32) |

**Tabulka 7.1:** *Formát paketů*

Aplikační kontrolní součet je počítán uvnitř ATMega32. Aplikační kontrolní součet je počítán z cílové a zdrojové ethernetové adresy, hodnoty EtherType a uživatelských dat. Ethernetový kontrolní součet je počítán, doplňován a kontrolován v ethernetovém adaptéru ENC28J60 a ATMega32 s ním nepracuje. Ethernetový kontrolní součet chrání paket před zanesením chyby při přenosu po ethernetu. Aplikační kontrolní součet chrání paket zejména při přenosu po sběrnici SPI mezi ENC28J60 a ATMega32.

Ethernetový adaptér ENC28J60 je možné nakonfigurovat i tak, že nebude kontrolní součet doplňovat sám, a bude namísto toho očekávat již připravený kontrolní součet v paketu předaném od ATMega32, a bude pouze kontrolovat tento připravený kontrolní součet. Teoreticky bych tedy mohl CRC-32 počítat na jednočipovém počítači ATMega32, a zabezpečit tak jedním kontrolním součtem jak SPI sběrnici, tak Ethernet. Nicméně výpočet CRC-32 je poměrně náročný na výpočetní prostředky, a tak jsem raději zvolil použití dodatečného kontrolního součtu.

- Uživatelská data každého paketu začínají operačním kódem.
  - Pro snadnější práci s pakety z pohledu člověka, jsou operační kódy inspirovány ASCII tabulkou.

- Stavy - `0x53` ('S' jako State)
  - Posílá ATMega32 na broadcast adresu
- Příkazy - `0x43` ('C' jako Command)
  - Je možné zaslat počítači ATMega32
  - Příkazy slouží k ovládání počítače ATMega32
- Dotazy - `0x51` ('Q' jako Query)
  - Je možné zaslat počítači ATMega32
  - Dotazy nemění stav počítače ATMega32
    - Slouží pouze ke zjištění informace (např. dump paměti)
- Potvrzení - `0x41` ('A' jako Acknowledgement)
  - Posílá ATMega32 jako reakci na příkazy ('C')
- Odpovědi - `0x52` ('R' jako Response)
  - Posílá ATMega32 jako reakci na dotazy ('Q')

V tabulce 7.2 jsou vyjmenovány všechny typy paketů, se kterými bootloader pracuje.

| Prefix | Další data | Popis |
|---|---|---|
| 0x53, 0x42 'SB' | Pořadí paketu - 0 až 15 (1 bajt) | Posílá bootloader 16x při bootování. Během této doby je možné přepnout bootloader do flashovacího režimu. |
| 0x43, 0x42 'CB' | - | Reboot - provede reset ATMega32. |
| 0x43, 0x51 'CQ' | - | Přeruší bootování a přepne bootloader do flashovacího režimu. Lze použít pouze při bootování. |
| 0x43, 0x57 'CW' | Číslo stránky (2 bajty), Nová data (128 bajtů) | Zápis nových dat do FLASH paměti na danou adresu dle čísla stránky. Lze použít pouze ve flashovacím režimu. |
| 0x51, 0x52 'QR' | Číslo stránky (2 bajty) | Čtení dat z FLASH paměti z adresy dle čísla stránky. Lze použít i za běhu aplikačního programu. |
| 0x51, 0x44 'QD' | - | Dump paměťového prostoru (registry + SRAM) |
| 0x41, 0x51 'AQ' | - | Bootloader úspěšně přepnut do flashovacího režimu. |
| 0x41, 0x4E, 0x51 'ANQ' | - | Bootloader se nepodařilo přepnout do flashovacího režimu (aplikační program již běží). |
| 0x41, 0x57 'AW' | Číslo stránky (2 bajty) | Nová data úspěšně zapsána do FLASH paměti |
| 0x41, 0x4E, 0x57 'ANW' | - | Nová data se do FLASH paměti nepodařilo zapsat (bootloader není ve flashovacím režimu). |
| 0x52, 0x44 'RD' | Počet stránek (1 bajt), Číslo stránky (1 bajt), Data (92 nebo 1024 bajtů) | Odpověď na dump paměťového prostoru (posláno ve více paketech - 1× registry, 2× SRAM) |
| 0x52, 0x52 'RR' | Číslo stránky (2 bajty), Data z FLASH (128 bajtů) | Odpověď na čtení dat z FLASH paměti |
| 0x41, 0x49 'AI' | - | Neplatný příkaz |
| 0x52, 0x49 'RI' | - | Neplatný dotaz |
| 0x49 'I' | Operační kód (1 bajt) | Neplatný operační kód |

**Tabulka 7.2:** *Typy paketů bootloaderu*

Příkaz 'QD' způsobí, že je přečten celý paměťový prostor včetně I/O registrů, což může v některých případech způsobit také škodu. Čtení některých I/O registrů má vedlejší efekt. Jedná se zejména o registr UDR (viz strana 141 datasheetu ATMega32 [1]), jehož čtení způsobí také vyčtení případných přijatých dat periferií USART, a může tak způsobit ztrátu těchto dat. Také čtení registru TCNT1L způsobí jako vedlejší efekt aktualizaci TCNT1H -

to by ale při běžném používání registru TCNT1 nemělo způsobit žádné problémy.

## 7.5 Start bootloaderu

### 7.5.1 Běžný start

1. Inicializace watchdogu
2. Inicializace ethernetového adaptéru ENC28J60
3. Zaslání 16 paketů 'SB' (viz tabulka 7.2) na broadcast adresu
4. Spuštění aplikačního programu

### 7.5.2 Příklad využití flashovacího režimu

1. Inicializace watchdogu
2. Inicializace ethernetového adaptéru ENC28J60
3. Zaslání nejvýše 16 paketů 'SB' (viz tabulka 7.2) na broadcast adresu
4. Přijetí paketu 'CQ' (viz tabulka 7.2)
5. Vstup do flashovacího režimu
6. Aktualizace aplikačního programu pomocí příkazů 'CW'
7. Dodatečná kontrola aplikačního programu pomocí příkazů 'QR'
8. Reboot do nového aplikačního programu pomocí příkazu 'CB'

Viz také kapitola 9.4.3 pro ukázku procesu flashování z pohledu protistrany.

## 7.6 API

Bootloader poskytuje aplikačnímu programu některé funkce prostřednictvím API. Funkce poskytované bootloaderem aplikačnímu programu, mají prefix „`b_`“, aby je bylo možné snadno odlišit.

- `b_receive` - Funkce pro přijímání ethernetových paketů
  - Při zavolání `b_receive` se krom navrácení případného přijatého paketu aplikačnímu programu také přijatý paket testuje, zda není určen pro bootloader (`EtherType = 0x0A01`). Pokud je paket určen bootloaderu, je bootloaderem zpracován.
  - Volání funkce `b_receive` také resetuje watchdog. Aplikační program musí volat funkci `b_receive` minimálně jednou za 2 sekundy.
  - Funkce `b_receive` vrací cílovou ethernetovou adresu, zdrojovou ethernetovou adresu, EtherType a uživatelská data. Volitelně může také `b_receive` vrátit informaci o tom, zda paket prošel kontrolou CRC-16-CCITT.

- `b_send` - Odesílání ethernetových paketů definovaných cílovou ethernetovou adresou, EthetType, vektorem paměťových oblastí a volitelným doplněním kontrolního součtu CRC-16-CCITT. Zdrojová ethernetová adresa a případný kontrolní součet jsou doplněny.
- `b_sendc` - Odesílání ethernetových paketů definovaných cílovou ethernetovou adresou, EtherType, jednou souvislou paměťovou oblastí a s přidáním kontrolního součtu CRC-16-CCITT. Zdrojová ethernetová adresa a kontrolní součet jsou doplněny.

Deklarace funkcí poskytovaných bootloaderem aplikačnímu programu se nachází v souboru `bfuncs.h`.

Při kompilaci bootloaderu je krom souborů určených k flashování vytvořen také soubor `bootloader_b`, který obsahuje vyexportované symboly s prefixem „`b_`“. Exportované symboly je možné vypsat například pomocí příkazu objdump. Ukázka:

```
$ avr-objdump -t bootloader_b

bootloader_b:     file format elf32-avr

SYMBOL TABLE:
00007000 l    d  .text  00000000 .text
000078a4 g     F .text  00000050 b_sendc
0000784c g     F .text  00000058 b_send
000077e2 g     F .text  0000006a b_receive
000077de g     F .text  00000004 b_check
```

Soubor `bootloader_b` je možné přímo použít při linkování. Příklady:

```
$ avr-gcc -mmcu=atmega32 -Wl,-R../bootloader/bootloader_b proj.c -o proj
$ avr-ld -R../bootloader/bootloader_b proj.o -o proj
```

# Kapitola 8

# Implementace aplikačního programu pro ATMega32

## 8.1 Komunikace

Aplikační program využívá pro komunikaci funkce bootloaderu `b_check`, `b_receive` a `b_sendc`. Formát paketů je stejný jako o bootloaderu, pouze EtherType je `0x0A02` namísto `0x0A01`. Konvence operačních kódů je podobná jako v případě bootloaderu. Používají se prefixy '`S`' a '`C`' (viz kapitola 7.4).

V tabulce 8.1 jsou vyjmenovány typy paketů, které implementují oba jednočipové počítače (č.1 i č.2).

| Prefix | Další data | Popis |
|---|---|---|
| `0x53, 0x54 'ST'` | Fixed-point hodnota unixového času s přesností na $\frac{1}{256}s$ (8 bajtů), Posun lokálního času vůči unixovému v sekundách (2 bajty) | Posílá aplikační program na jednočipových počítačích pro účely synchronizace času. |
| `0x43, 0x46 'CF'` | Fixed-point hodnota unixového času s přesností na $\frac{1}{256}s$ (8 bajtů) | Posílá přístupový server pro vynucení vlastního času pro účely synchronizace času. |
| `0x41, 0x46 'AF'` | Fixed-point hodnota unixového času s přesností na $\frac{1}{256}s$ (8 bajtů) | Posílá jednočipový počítač jako odpověď na paket 'CF'. |
| `0x43, 0x53 'CS'` | Perioda časovače v osminásobcích hodinového signálu (2 bajty) | Posílá přístupový server pro doladění periody časovače pro účely synchronizace času. |
| `0x41, 0x53 'AS'` | Perioda časovače v osminásobcích hodinového signálu (2 bajty) | Posílá jednočipový počítač jako odpověď na paket 'CS'. |

| Prefix | Další data | Popis |
|---|---|---|
| `0x43, 0x5A 'CZ'` | Posun lokálního času vůči unixovému v sekundách (2 bajty) | Posílá přístupový server pro účely nastavení aktuálního časového pásma. |
| `0x41, 0x5A 'AZ'` | Posun lokálního času vůči unixovému v sekundách (2 bajty) | Posílá jednočipový počítač jako odpověď na paket 'CZ'. |
| `0x43, 0x4D 'CM'` | Relé ON maska (2 bajty), Relé OFF maska (2 bajty), Maska masek (2 bajty) | Je možné použít pro manuální převzetí kontroly nad relé připojenými k jednočipovému počítači. |
| `0x41, 0x4D 'AM'` | Relé ON maska (2 bajty), Relé OFF maska (2 bajty) | Posílá jednočipový počítač jako odpověď na paket 'CM'. |
| `0x53, 0x50 'SP'` | Stav relé po vymaskování (2 bajty), Aplikačním programem řízený stav relé (2 bajty), Relé ON maska (2 bajty), Relé OFF maska (2 bajty), Příznaky (2 bajty), Stav registrů PORTA, PORTB, PORTC, PORTD, PINA, PINB, PINC, PIND v tomto pořadí (8 bajtů) | Zasílá periodicky jednočipový počítač pro účely ladění a logování stavu. |
| `0x41, 0x49 'AI'` | Kód příkazu (1 bajt) | Neplatný příkaz |
| `0x49 'I'` | Operační kód (1 bajt) | Neplatný paket (neznámý operační kód) |

**Tabulka 8.1:** *Typy paketů aplikačního programu*

## 8.2 Čas

Oba jednočipové počítače potřebují nějakým způsobem pracovat s časem. Pomocí času je řízeno například čtení senzorů, pravidelné zasílání paketů o stavu jednočipového počítače. Počítač č.1 používá čas také například ke zjištění denní doby a řídí podle něj osvětlení. Pro udržení přesného času využívám vnitřní periferii Timer/Counter1 vestavěnou do jednočipového počítače ATMega32. Timer/Counter1 je 16 bitový čítač/časovač. Používám ho v režimu Clear Timer on Compare Match (CTC) - WGM1 = 4 (WGM10 = 0, WGM11 = 0, WGM12 = 1, WGM13 = 0). Přehled režimů periferie Timer/Counter1 lze nalézt na straně 109 v datasheetu ATMega32 [1]. Podrobnosti o režimu CTC lze v tomto datasheetu nalézt na straně 98.

Princip použití periferie Timer/Counter1 v režimu CTC je následující. Timer/Counter1 běží na pozadí, řízen hlavním hodinovým signálem jednočipového počítače. To znamená, že nezávisle na tom, co počítač provádí za instrukce, časovač bude narůstat stále stejně rychle. Ve chvíli, kdy časovač přeteče (dočítá do hodnoty `OCR1A`), nastaví příznak `OCF1A`. Příznak `OCF1A` se pravidelně kontroluje, a když je zjištěno, že je nastaven, tak se resetuje a zvýší se časová proměnná nesoucí informaci o aktuálním čase. Obsluhu příznaku `OCF1A` a

navyšování časové proměnné zajišťuje plánovač úloh (viz kapitola 8.3).

Kontrola, zda nedošlo k nastavení příznaku `OCF1A` se tedy musí provádět dostatečně často, aby v době mezi kontrolami nedošlo k více než jednomu přetečení časovače. To znamená, že perioda přetečení časovače musí být vyšší než nejdelší úloha, kterou bude jednočipový počítač mezi kontrolami provádět. Významně dlouhé souvislé úlohy, které je potřeba provádět, jsou:

- Komunikace s teplotním senzorem DS18B20, která může trvat až přibližně $2.6\,ms$ (12 bitová konverze teploty může teplotnímu senzoru DS18B20 trvat až $750\,ms$, ale na tuto konverzi počítač nečeká a nechá ji provádět, zatímco se věnuje jiným činnostem).
- Měření vzdálenosti senzorem HC-SR04, které může trvat až přibližně $3.7\,ms$ (HC-SR04 je schopno podle specifikace výrobce [2] měřit vzdálenosti až $4\,m$, což odpovídá době měření přibližně $25\,ms$, nicméně pro potřeby měření vody v nádrži limituji měření na přibližně $4\,dm$, delší vzdálenosti měřit nepotřebuji, pokud by měření trvalo delší dobu, než ekvivalent $4\,dm$, považuji měření za chybné).

Abych mohl plánovat úlohy v jednočipovém počítači přesněji, posunul jsem časovou proměnnou o bajt. Díky tomu může časová proměnná nést informaci o čase s přesností na $\frac{1}{256}\,s$ (přibližně $3.9\,ms$). Získat z proměnné sekundy je přitom velmi efektivní (stačí zaadresovat o bajt výše).

Vnitřní periferii Timer/Counter1 je tedy potřeba nakonfigurovat tak, aby došlo k přetečení jednou za $\frac{2^n}{256}\,s$, kde $n \in \mathbb{N}_0$. Časová proměnná se pak bude zvyšovat při každém přetečení o $\frac{2^n}{256}\,s$. Díky tomu, že se časová proměnná navyšuje o mocninu dvojky, bude každá sekunda začínat celočíselnou hodnotou na pozici sekund a každá sekunda reprezentovaná časovou proměnnou bude mít stejnou skutečnou délku. Perioda periferie Timer/-Counter1 v režimu CTC se konfiguruje především pomocí registru `OCR1A`, který určuje, do které hodnoty časovač čítá, a bitů `CS10`, `CS11`, `CS12`, kterými lze konfigurovat předdělička pro Timer/Counter1.

Při volbě `CS1`=1 (bez předdělička), bude časovač narůstat přesně s hodinovým signálem. Počítače ATMega32 taktuji na frekvenci $16\,MHz$. Abych dosáhl periody $\frac{1}{256}\,s$, potřebuji tuto frekvenci vydělit hodnotou 62500 ($= \frac{16000000}{256}$). Pokud tedy nastavím `OCR1A` na hodnotu 62500, dosáhnu periody $\frac{1}{256}\,s$.

Taková konfigurace není úplně vhodná pro mé potřeby. Přibližně $3.9\,ms$ by sice měla být dostačující perioda, aby počítač ATMega32 stihl každou úlohu, kterou je potřeba souvisle provést, ale rezerva pouhých $0.2\,ms$, je velmi málo. Je potřeba také počítat s nějakou režií před spuštěním úlohy, po dokončení úlohy, režii bootloaderu, případně může být také potřeba zpracovat nějaký příchozí ethernetový paket.

Zvýším tedy předdělička na `CS1`=2 (dělení /8). A hodnotu `OCR1A` snížím na polovinu (31250). Tím se perioda zvýší na $\frac{4}{256}\,s$ (přibližně $15.6\,ms$), což je dostatečná časová rezerva. Registr `OCR1A` je přibližně v polovině své maximální velikosti, což představuje optimální přesnost v kombinaci s možností zpomalení času až o více než 100 %, což by nemělo být nikdy potřeba. S každým přetečením časovače si jednočipový počítač zvýší časovou proměnnou o hodnotu 4. Časové proměnné si jednočipové počítače udržují dvě (`ticks` a `time`).

Proměnná `ticks` představuje počet $\frac{1}{256}\,s$ od resetu (bootu) jednočipového počítače. V rámci běhu aplikačního programu se může pouze zvyšovat a je vhodná pro plánování periodických událostí (např. pravidelné oznamování stavu jednou za 8 sekund). Na této proměnné je založen plánovač úloh (viz kapitola 8.3).

Proměnná `time` představuje fixed-point hodnotu unixového času s přesností na $\frac{1}{256}\,s$. Proměnná `time` se může při větší desynchronizaci času i snížit (viz kapitola 8.2.1). Proměnná

`time` je vhodná pro plánování událostí v rámci denní doby (např. rozsvícení osvětlení ráno nebo zhasnutí večer).

### 8.2.1 Synchronizace času

Aby fungovaly funkce závislé na reálném čase (např. řízení osvětlení), je potřeba na jednočipovém počítači udržovat reálný čas. Jednočipový počítač ATMega32 není schopen čas udržet sám. Přesnost krystalového oscilátoru je pro účely udržení reálného času nedostatečná (nevím, jakou toleranci mají krystaly na deskách JY-MEGA32, ale experimentálně jsem zjistil, že jejich odchylka od skutečného času je přibližně $-64\,\mu Hz/Hz$ - to znamená, že například po půl roce, by vzniklo čtvrthodinové zpoždění). Nepřesnost krystalového oscilátoru by bylo možné kompenzovat, nicméně dá se předpokládat, že se odchylka krystalu od skutečného času bude také postupem času měnit. Dalším problémem je, že při výpadku napájení nemá jednočipový počítač jak udržet čas. Z praktických důvodů se tedy dá vyloučit manuální seřizování času na jednočipových počítačích a je nutné čas synchronizovat automaticky. Pro účely automatické synchronizace času využívám přístupový server, který je připojený prostřednictvím ethernetu a na kterém je čas již synchronizován pomocí NTP.

Jednočipový počítač periodicky každých 8 sekund oznamuje přístupovému serveru svůj aktuální čas pomocí paketů '`ST`'. Přístupový server spočítá odchylku od svého aktuálního času. Pokud je tato odchylka větší než stanovený limit (mám nastaveno na 256 - tedy $1s$), přístupový server reaguje paketem '`CF`', s parametrem aktuálního času, kterým vynutí změnu času na konkrétní absolutní hodnotu. Pokud jednočipový počítač přijme paket '`CF`', nastaví si aktuální časovou proměnnou `time` podle hodnoty v tomto paketu a potvrdí nastavení paketem '`AF`'. Pokud tato odchylka nepřesahuje stanovený limit, přístupový server reaguje paketem '`CS`' s parametrem součtu rozdílu časů a výchozí periody časovače 32148 (používám o dvě nižší hodnotu než původní vypočtenou hodnotu 32150, protože jsem experimentálně zjistil, že je přesnější). Pokud jednočipový počítač přijme paket '`CS`', nastaví si hodnotu parametru z tohoto paketu do registru `OCR1A` a nastavení potvrdí paketem '`AS`'. Pro formáty paketů '`ST`', '`CS`', '`CF`', '`AS`' a '`AF`' viz tabulka 8.1.

Díky tomuto přístupu jsou velké odchylky od reálného času (např. po resetu jednočipového počítače) ihned opraveny. Při nižších odchylkách způsobených nepřesností oscilátoru se pouze dolaďuje perioda časovače a tím se čas na jednočipovém počítači mírně zrychluje nebo zpomaluje. K opravě času tak dojde bez náhlé změny časové proměnné. Toto řešení dolaďování periody časovače není úplně ideální, protože při odchylce oscilátoru od výchozí hodnoty 32148 zůstane čas na jednočipovém počítači stabilně posunutý o právě tuto odchylku. Jednoduchou integrací této odchylky by se dal tento posun snadno kompenzovat, nicméně úloha jednočipových počítačů nevyžaduje takovou přesnost času, aby to bylo potřeba.

Na přístupovém serveru synchronizaci času obsluhuje služba `erc_timesync` - viz kapitola 9.3.2.

### 8.2.2 Posun času (standardní a letní čas)

Z praktických důvodů je žádoucí, aby řízení osvětlení fungovalo v souladu s posunem času (střídání SEČ a SELČ). To znamená, že rozsvěcení a zhasínání by se mělo provádět ve stanovené hodině dle aktuálního lokálního času a ne přímo na základě unixového času. Zjištění, jaké je aktuálně časové pásmo na základě unixového času je reálné, ale poměrně složité. Než vyhodnocovat, zda je posunutý čas, na jednočipových počítačích, jsem raději zvolil synchronizaci této informace z přístupového serveru.

Jednočipové počítače zasílají v synchronizačních paketech ('ST') krom svého aktuálního času také aktuální časový posun vůči unixovému času v sekundách (tedy za běžných okolností buď hodnotu 3600 nebo 7200). Pokud přístupový server detekuje, že se časový posun na jednočipovém počítači liší od aktuálního časového posunu na přístupovém serveru, zašle jednočipovému počítači paket typu 'CZ', kterým nastaví správný posun času. Pro formát paketu 'CZ' viz tabulka 8.1.

K posunu času tedy na jednočipovém počítači nedochází přesně ve správný okamžik, nicméně pro potřeby řízení osvětlení je toto řešení dostačující.

Na přístupovém serveru synchronizaci posunu času obsluhuje služba `erc_timesync` - viz kapitola 9.3.2.

## 8.3 Plánování úloh

Pro účely plánování úloh jsem implementoval jednoduchý plánovač, který pracuje s proměnnou `ticks` (více o proměnné `ticks` v kapitole 8.2). Plánovač úloh obsluhuje periferii Timer/Counter1 a zajišťuje navyšování proměnné `ticks` i resetování příznaku `OCF1A`. Plánovač zajišťuje spouštění naplánovaných úloh a také spouštění obslužných funkcí reagujících na příkazy přijaté prostřednictvím ethernetu. Plánovač při spuštění očekává, že je mu předán seznam úloh, které má periodicky spouštět, a seznam obslužných funkcí pro jednotlivé typy ethernetových paketů.

Záznam v seznamu úloh nese krom obslužné funkce úlohy také informaci o plánování této úlohy. Plánování je definováno formou masky (`mask`) a porovnání (`cmp`). Dojde-li k přetečení časovače a tedy změně proměnné `ticks`, plánovač postupně projde všechny úlohy v seznamu úloh a spustí každou úlohu, která je na aktuální hodnotu `ticks` naplánována. Zda je úloha na aktuální hodnotu `ticks` naplánována se zjistí tak, že se hodnota proměnné `ticks` vymaskuje (bitový součin / and) pomocí `mask` a výsledek se porovná s hodnotou `cmp`. Pokud dojde ke shodě, znamená to, že je úloha na aktuální hodnotu `ticks` naplánována a je potřeba ji spustit. Maska tedy definuje periodu spouštění úlohy a hodnota `cmp` definuje fázový posun v rámci periody. Na danou hodnotu `ticks` může být naplánováno i více úloh, v takovém případě se spustí všechny, a to dle pořadí v jakém jsou v seznamu naplánovaných úloh.

### Příklad

| Úloha | `mask` | `cmp` |
|---|---|---|
| Task1 | 0x3FF | 0x040 |
| Task2 | 0x1FF | 0x0C0 |

**Tabulka 8.2:** *Příklad konfigurace plánovače*

Pokud budou naplánovány úlohy podle tabulky 8.2, bude perioda úlohy Task1 1024 ticks (tedy 4 $s$) a perioda úlohy Task2 512 ticks (tedy 2 $s$).

Spouštěny budou úlohy následně:

(0.25$s$ Task1), (0.75$s$ Task2), (2.75$s$ Task2), (4.25$s$ Task1), (4.75$s$ Task2), (6.75$s$ Task2), (8.25$s$ Task1), (8.75$s$ Task2), (10.75 Task2), (12.25$s$ Task1) ...

Pomocí tohoto plánovače je tedy poměrně snadné naplánovat úlohy tak, aby nikdy nedošlo k nečekanému souběhu více úloh naráz a nežádoucímu přetečení časovače během vykonávání úloh.

Pro konkrétní konfigurace plánovače na jednočipových počítačích viz tabulky 8.3 a 8.5.

## 8.4 Maskování relé

Při vývoji vestavěného systému i například při doplňování dalších funkcí je často užitečné mít možnost manuálně ovládat relé připojené k jednočipovým počítačům, aniž by bylo nutné upravovat firmware. Pro tyto účely jsem implementoval možnost maskování relé. Při provozu aplikační program modifikuje 16 bitovou proměnnou `relay.app`, která nese informaci o tom, jaká relé mají být dle aplikačního programu sepnuta. K této proměnné je následně bitově přičtena (`or`) proměnná `relay.mask_on` a následně je proveden bitový součin (`and`) s proměnnou `relay.mask_off`. Výsledek se zapíše do proměnné `relay.masked` a na základě proměnné `relay.masked` se nastavují relé. Za běžných okolností jsou proměnné `relay.mask_on` a `relay.mask_off` nulové, takže platí `relay.masked` = `relay.app`. V případě potřeby je možné nastavit proměnné `relay.mask_on` a `relay.mask_off` a tím převzít kontrolu nad požadovanými relé.

Nastavování maskovacích proměnných se provádí pomocí paketu 'CM'. Pro účely ladění a logování stavu jednočipové počítače periodicky zasílají pakety 'SP', ve kterých je mimo jiné také aktuální konfigurace proměnných `relay.mask_on` a `relay.mask_off`. Pro formát paketů 'CM' a 'SP' viz tabulka 8.1.

## 8.5 Jednočipový počítač č.1

Jednočipový počítač č.1 řídí osvětlení, větrání a obměnu vody. Tyto tři činnosti jsem rozdělil do třech úloh v plánovači: `light_control`, `fan_control` a `water_renewal`.

### 8.5.1 Konfigurace plánovače

V tabulce 8.3 jsou znázorněny úlohy naplánované na jednočipovém počítači č.1.

| Úloha | mask | cmp | Perioda | Popis |
|---|---|---|---|---|
| time_advance | 0x000 | 0x000 | $\frac{1}{64}\,s$ | zvyšuje proměnnou time |
| time_sync_req | 0x7FC | 0x000 | $8\,s$ | zasílá pakety 'ST' |
| status_reports | 0x7FC | 0x010 | $8\,s$ | zasílá pakety 'SP' |
| switch_handler | 0x0FC | 0x020 | $1\,s$ | reaguje na stavy vypínačů |
| light_control | 0x0FC | 0x030 | $1\,s$ | ovládá osvětlení |
| fan_control | 0xFFC | 0x030 | $1\,s$ | ovládá větrání |
| water_renewal | 0xFFC | 0x040 | $16\,s$ | ovládá obměnu vody (vypouštění) |
| setrelays | 0x0FC | 0x040 | $1\,s$ | maskuje a nastavuje relé |
| water_renewal_report | 0xFFC | 0x050 | $16\,s$ | zasílá pakety 'SN' |

**Tabulka 8.3:** *Konfigurace plánovače jednočipového počítače č.1*

### 8.5.2 Komunikace

Jednočipový počítač č.1 podporuje krom již zmíněných typů ethernetových paketů v tabulce 8.1 také některé další typy ethernetových paketů v souvislosti s konkrétními úlohami, které jednočipový počítač č.1 provádí. Tyto další podporované typy ethernetových paketů jsou znázorněny v tabulce 8.4.

| Prefix | Další data | Popis |
|---|---|---|
| `0x53, 0x57 'SN'` | Stav vypouštění (1 bajt), Aktivní ventil (1 bajt), Stav žádosti o uzamčení nádrže (1 bajt), Stav uzamčení nádrže (2 bajty), Počet ventilů (1 bajt), Stavy jednotlivých ventilů (2 bajty na každý ventil) | Zasílá pravidelně jednočipový počítač pro účely ladění a logování stavu obměny vody. |
| `0x43, 0x47 'CG'` | Nové stavy příznaků (2 bajty), Maska (2 bajty) | Požadavek na změnu příznaků. |
| `0x41, 0x47 'AG'` | Nové stavy příznaků (2 bajty) | Posílá jednočipový počítač v reakci na paket 'CG'. |
| `0x43, 0x56 'CV'` | Sčítanci ke stavům ventilů (2 bajty na každý ventil) | Požadavek na přičtení hodnoty k ventilům pro případné manuální úpravy obměny vody. |
| `0x41, 0x56 'AV'` | Výsledky po přičtení ke stavům ventilů a původní sčítanci (4 bajty na každý ventil) | Posílá jednočipový počítač v reakci na paket 'CV'. |

**Tabulka 8.4:** *Další typy paketů, se kterými pracuje jednočipový počítač č.1*

Jednočipový počítač také dále pracuje s pakety typu '`CL`', '`AL`' a '`CU`', pomocí kterých komunikuje přímo s jednočipovým počítačem č.2. Pro formát těchto paketů viz tabulka 8.6.

### 8.5.3 Příznaky

Počítač č.1 využívá příznaky. Jde o jednobitové hodnoty, které lze nastavovat pomocí paketů typu '`CG`' (pro formát paketu '`CG`' viz tabulka 8.4). Příznaky jsou:

- `FLAG_LIGHT` pro ovládání světel. Příznak `FLAG_LIGHT` je ovládán také vypínačem. Pomocí příznaku `FLAG_LIGHT` je možné rozsvítit v odchovně i v případě, že by jinak bylo zhasnuto. Na příznak `FLAG_LIGHT` reaguje úloha `light_control`.
- `FLAG_MASK_VALVE`x pro ovládání vypouštění. Příznaky `FLAG_MASK_VALVE`x jsou ve výchozím stavu zapnuty. Jejich vypnutím je možné vypnout vypouštění zvolenými ventily. Zakázat vypouštění zvolenými ventily je možné i pomocí maskování relé (`relay.mask_off`), ale při použití maskování relé na rozdíl od příznaků se nadále bude získávat zámek nádrže (včetně případného ohřevu), i když není potřeba a budou se snižovat naakumulované hodnoty ventilů, jako kdyby docházelo k vypouštění, přestože k žádnému vypouštění nedochází.

### 8.5.4 Osvětlení (úloha light_control)

Úloha `light_control` řídí ovládání hlavních i tlumených světel. Úloha light_control reaguje na příznak `FLAG_LIGHT`, stav senzoru světla a aktuální čas. Pozvolné rozsvěcení a zhasínání je řešeno pomocí stavové proměnné `light_ctrls.state`. V případě, že má být rozsvíceno, proměnná `light_ctrls.state` se zvyšuje. Má-li být zhasnuto, proměnná `light_ctrls.state` se snižuje. Na základě hodnoty této proměnné se ovládají hlavní i tlumená světla. Na obrázku 8.1 je znázorněna závislost svitu světel na hodnotě proměnné `light_ctrls.state`.


**Obrázek 8.1:** *Svit hlavních (spodní žlutá) a tlumených (horní oranžová) světel v závislosti na proměnné* `light_ctrls.state`

Pokud je například ráno před rozsvěcením, hodnota proměnné `light_ctrls.state` je 0. Jakmile nastane čas, kdy má dojít k rozsvícení (9:00), proměnná `light_ctrls.state` se začne zvyšovat. Hned po sekundě dojde k rozsvícení tlumeného osvětlení. Po 300 $s$ (5 minut) dojde k rozsvícení hlavního osvětlení. Za dalších 10 sekund zhasnou tlumená světla.

V případě, že rozsvěcí obsluha (nastaven příznak `FLAG_LIGHT`, např. zapnutím vypínače nebo přes uživatelskou aplikaci), proměnná `light_ctrls.state` se zvyšuje rychleji, aby rozsvěcení netrvalo moc dlouho. Pokud je den a světla jsou zhasnuta z důvodu dostatečného venkovního osvětlení (dle stavu senzoru světla) a obsluha rozsvítí příznakem `FLAG_LIGHT`, hlavní světla se rozsvítí ihned.

### 8.5.5 Větrání (úloha fan_control)

Větrání je zřejmě nejjednodušší úloha, kterou jednočipový počítač č.1 vykonává. Pro účely větrání si jednočipový počítač udržuje proměnnou `fan_ctrls.ticks`, kterou inkrementuje každé spuštění úlohy `fan_control` (tedy každých 16 $s$, viz konfigurace plánovače jednočipového počítače č.1 - tabulka 8.3). Zda má být ventilátor zapnutý, se pak zjistí porovnáním proměnné `fan_ctrls.ticks` s hodnotou pracovní hranice `0x40` pro vnitřní ventilátor a hodnotou `0x16` pro vnější ventilátor. Pokud je hodnota `fan_ctrls.ticks` pod danou pracovní hranicí, ventilátor je zapnut. Proměnná `fan_ctrls.ticks` je typu `uint8_t`, takže po dosažení hodnoty 256 přeteče a celý cyklus se opakuje. Perioda cyklu je tedy 4096 $s$.

Výsledkem je, že pro vnitřní ventilátor se opakuje cyklus 1024 $s$ zapnuto, 3072 $s$ vypnuto. U vnějšího ventilátoru je to 352 $s$ zapnuto a 3744 $s$ vypnuto. Střída vnitřního ventilátoru je tedy 25 % a střída vnějšího ventilátoru je přibližně 8.6 %, což odpovídá požadavkům (viz kapitola 2.4).

### 8.5.6 Obměna vody (úloha water_renewal)

Proces odměny vody spočívá v tom, že je pro každý vypouštěcí ventil udržována a navyšována (akumulována) hodnota udávající množství vody, které je potřeba obměnit (ve formě času). Na každé naplánování úlohy `water_renewal` je tato hodnota zvýšena o 1. V případě, že množství vody, které je potřeba obměnit, přesáhne stanovenou mez (nastaveno na 64 $s$ vypouštěcího času), je obměna provedena. Před provedením obměny vody je nejdřív od jednočipového počítače č.2 vyžádáno uzamčení nádrže s připravenou vodou ohřátou na

požadovanou teplotu. Jednočipovému počítači č.2 jsou zasílány pakety typu 'CL', dokud není obdržena odpověď o úspěšném uzamčení nádrže (pro formát paketu 'CL' viz tabulka 8.6). Samotné provedení obměny vody je pak realizováno otevřením příslušného vypouštěcího ventilu na potřebnou dobu.

Během obměny se pro příslušný ventil snižuje naakumulovaná hodnota udávající množství vody, které je potřeba obměnit. Snižování naakumulované hodnoty je prováděno po 72 (tedy 72 krát rychleji než akumulace). Rychlost snižování naakumulované hodnoty přímo ovlivňuje množství obměněné vody za jednotku času. Při rychlosti 72 například na 1 den akumulace připadá 20 minut vypouštěcího času, což odpovídá požadovanému množství obměněné vody za jednotku času (viz konec kapitoly 5.4).

Aby nemohlo dojít k vyčerpání celé nádrže, je uzamčení nádrže omezeno na stanovený čas (nastaveno na přibližně 23 minut). Po vypršení tohoto času je obměna vody přerušena a nádrž uvolněna i v případě, že je nadále potřeba vodu obměňovat.

Při vypouštění vody se automaticky spouští také čerpadlo, které čerpá vodu z nádrže do rozvodu. Toto čerpadlo je poměrně hlučné, a proto jsem do úlohy `water_renewal` implementoval také noční (tichý) režim. Noční režim je nastaven od 21:00 do 8:00. Během nočního režimu je hranice naakumulované hodnoty, při které se začne počítač pokoušet o obměnu vody, zvýšena na přibližně 23 $h$ akumulačního času. Díky nočnímu režimu se za běžných okolností v noci voda neobměňuje vůbec. Pouze pokud by se nestihlo obměnit dostatečné množství vody přes den (například pokud by v zimním období byla teplota vody z vodních zdrojů tak nízká, že by ji přímotopný ohřívač nestíhal dostatečně rychle ohřívat), tak začne jednočipový počítač obměňovat vodu i v nočních hodinách.

## 8.6 Jednočipový počítač č.2

Jednočipový počítač č.2 měří teplotu v nádrži pomocí teplotního senzoru DS18B20 (o DS18B20 viz kapitola 4.1.1), měří výšku hladiny pomocí dvojice senzorů HC-SR04 (o HC-SR04 viz kapitola 4.2.3) a řídí dopouštění nádrže a ohřev vody. Tyto činnosti jsem rozdělil do třech úloh v plánovači: `ds18b20_handler`, `hcsr04_handler` a `watertank_control`.

### 8.6.1 Konfigurace plánovače

V tabulce 8.5 jsou znázorněny úlohy naplánované na jednočipovém počítači č.2.

| Úloha | `mask` | `cmp` | Perioda | Popis |
|---|---|---|---|---|
| `time_advance` | 0x000 | 0x000 | $\frac{1}{64}\,s$ | zvyšuje proměnnou `time` |
| `time_sync_req` | 0x7FC | 0x000 | $8\,s$ | zasílá pakety 'ST' |
| `status_reports` | 0x7FC | 0x010 | $8\,s$ | zasílá pakety 'SP' |
| `water_status_reports` | 0x7FC | 0x020 | $1\,s$ | zasílá pakety 'SW' |
| `watertank_control` | 0x0FC | 0x040 | $1\,s$ | ovládá ohřev a dopouštění |
| `setrelays` | 0x0FC | 0x040 | $1\,s$ | maskuje a nastavuje relé |
| `ds18b20_handler` | 0x0FC | 0x050 | $1\,s$ | měří teplotu |
| `hcsr04_handler` | 0x3FC | 0x060 | $4\,s$ | měří výšku hladiny |

**Tabulka 8.5:** *Konfigurace plánovače jednočipového počítače č.2*

### 8.6.2 Komunikace

Jednočipový počítač č.2 podporuje krom již zmíněných typů ethernetových paketů v tabulce 8.1 také některé další typy ethernetových paketů v souvislosti s konkrétními úlohami, které jednočipový počítač č.2 provádí. Tyto další podporované typy ethernetových paketů jsou znázorněny v tabulce 8.6.

| Prefix | Další data | Popis |
|---|---|---|
| `0x53, 0x57 'SW'` | Počet nedávných selhání měření výšky hladiny (1 bajt), Výška hladiny (1 bajt), Stav plnění nádrže (1 bajt), Stav naplnění nádrže (1 bajt), Stav ohřevu vody (1 bajt), Stav měření teploty (1 bajt), Stav uzamčení nádrže (1 bajt), Nastavená minimální teplota pro ohřev (2 bajty), Čas do vypršení zámku nádrže (2 bajty) | Zasílá pravidelně jednočipový počítač pro účely ladění a logování stavu a řízení nádrže s vodou. |
| `0x53, 0x48, 0x53 'SHS'` | Id teplotního senzoru (1 bajt), Fixed-point teplota s přesností na $\frac{1}{16}$ °C (2 bajty) | Posílá jednočipový počítač při úspěšném měření teploty. |
| `0x53, 0x48, 0x46 'SHF'` | Id teplotního senzoru (1 bajt) | Posílá jednočipový počítač při nezdařeném měření teploty. |
| `0x53, 0x53 'SS'` | Naměřená hodnota senzoru 1 (1 bajt), Naměřená hodnota senzoru 2 (1 bajt), Počet nedávných selhání měření (1 bajt), Vypočtená průměrná výška hladiny (1 bajt) | Posílá jednočipový počítač po měření výšky hladiny. |
| `0x43, 0x4C 'CL'` | Minimální požadovaná teplota (2 bajty), Maximální požadovaná teplota (2 bajty) | Požadavek na uzamčení nádrže se stanovenou teplotou |
| `0x41, 0x4C 'AL'` | Stav připravenosti nádrže (1 bajt), Stav uzamčení nádrže (1 bajt) | Posílá jednočipový počítač v reakci na paket 'CL'. |
| `0x43, 0x55 'CU'` | - | Požadavek na uvolnění nádrže |
| `0x41, 0x55 'AU'` | - | Posílá jednočipový počítač v reakci na paket 'CU'. |

**Tabulka 8.6:** *Další typy paketů, se kterými pracuje jednočipový počítač č.2*

### 8.6.3 Měření teploty (úloha ds18b20_handler)

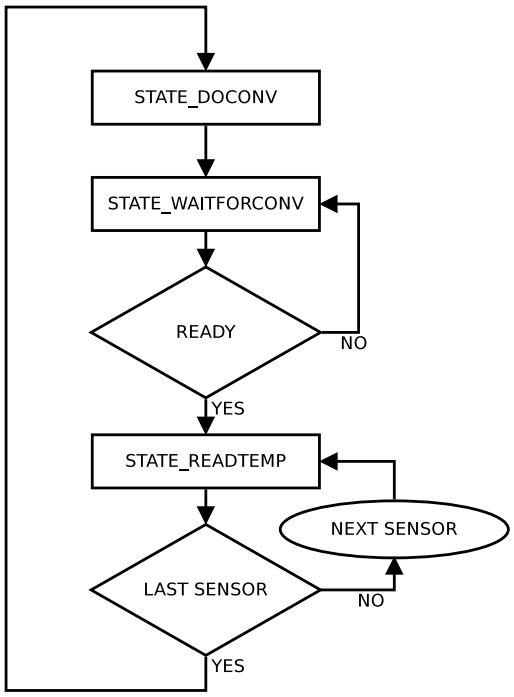
V budoucnu je v plánu měřit teplotu také na počítači č.1, nicméně prozatím měření teploty provádí pouze počítač č.2, a to pomocí jednoho teplotního senzoru DS18B20 ve vodotěsném pouzdře umístěného v nádrži (o DS18B20 viz kapitola 4.1.1). Současná implementace `ds18b20_handler` již počítá s měřením pomocí více senzorů DS18B20 připojených na společnou 1-Wire sběrnici (o 1-Wire viz kapitola 3.2). Měření teploty je rozděleno do třech stavů: `STATE_DOCONV`, `STATE_WAITFORCONV` a `STATE_READTEMP`, přičemž na jedno naplánování úlohy `ds18b20_handler` je vykonávána pouze část měření dle aktuálního stavu.

Ve stavu `STATE_DOCONV` se všem senzorům DS18B20 připojeným na sběrnici zašle příkaz `Convert T` (pro příkazy podporované senzorem DS18B20 a jejich významy viz strana 12 datasheetu [5]). Po zaslání příkazu `Convert T` se pro další naplánování úlohy `ds18b20_handler` nastaví stav `STATE_WAITFORCONV`.

Ve stavu `STATE_WAITFORCONV` jednočipový počítač zjišťuje stav konverze teploty pomocí read time slotů. Pokud konverze stále probíhá, je přečtena hodnota 0. Pokud je konverze na všech senzorech již dokončena, přečtena je hodnota 1. Toto chování senzorů DS18B20 je popsáno v sekci Operation—Measuring Temperature na straně 5 v datasheetu [5]. Pokud konverze stále není dokončena, stav `STATE_WAITFORCONV` zůstává zachován i pro další naplánování úlohy `ds18b20_handler`. Pokud je již konverze dokončena, stav se posune do poslední fáze `STATE_READTEMP`. Konverze teploty by měla senzoru DS18B20 trvat nejvýše $750ms$. Úloha `ds18b20_handler` je naplánována s periodou $1s$, tudíž by nemělo dojít k žádnému čekání na konverzi. Nicméně při připojení více senzorů DS18B20 může být žádoucí zkrátit periodu plánování úlohy `ds18b20_handler`, aby se urychlila celková doba čtení z jednotlivých senzorů v rámci fáze `STATE_READTEMP`. V takovém případě je stav `STATE_WAITFORCONV` nutný.

Ve stavu `STATE_READTEMP` se ze všech nakonfigurovaných senzorů DS18B20 čtou naměřené teploty pomocí příkazu `Read Scratchpad`. Přičemž na jedno naplánování úlohy `ds18b20_handler` se provádí čtení právě z jednoho senzoru DS18B20. Po dokončení čtení naměřené teploty se po ethernetu zasílá paket typu 'SHS' nebo 'SHF' pro účely logování a monitorování (pro formáty paketů 'SHS' a 'SHF' viz tabulka 8.6). Pokud je dokončeno čtení z posledního senzoru, měření teploty je dokončeno a cyklus se opakuje počínaje opět stavem `STATE_DOCONV`.

Obrázek 8.2 ilustruje přechody mezi stavy v rámci úlohy `ds18b20_handler`.




**Obrázek 8.2:** *Ilustrace přechodů mezi stavy úlohy* `ds18b20_handler`

### 8.6.4 Měření výšky hladiny (úloha hcsr04_handler)

Měření výšky hladiny je realizováno pomocí dvojice senzorů HC-SR04 (o HC-SR04 viz kapitola 4.2.3). Výrobce stanovuje rozsah měření na $4\,m$, čemuž odpovídá doba měření přibližně $25\,ms$. To je dáno rychlostí šíření zvuku ve vzduchu (přibližně $340\,m/s$). Vzdálenosti $4\,m$ pak odpovídá doba měření $2 \cdot \frac{4}{340} \doteq 23.5\,ms$ (podíl je potřeba násobit číslem 2 - pro cestu ultrazvukového signálu tam i zpět). Nějakou dobu také trvá vyslání ultrazvukového signálu a režie na straně jednočipového počítače. Pokud bych skutečně potřeboval měřit vzdálenosti do $4\,m$, musel bych zvolit větší periodu časovače Timer/Counter1 (viz kapitola 8.2), nebo měření nějakým způsobem rozdělit na více period časovače. Díky tomu, že potřebuji měřit vzdálenosti do $4\,dm$, mohu měření limitovat na přibližně $3.7\,ms$, což není problém provést v jedné periodě časovače. Pokud měření trvá déle, je to považováno za chybu. Jedno spuštění úlohy `hcsr04_handler` tedy provede jedno kompletní měření vzdálenosti pomocí jednoho senzoru HC-SR04.

Úloha `hcsr04_handler` si uchovává informaci o tom, kterým senzorem se měřilo naposled, a při každém spuštění senzory střídá. Po úspěšném provedení měření pomocí obou senzorů se vypočte odchylka měření obou senzorů. Pokud odchylka překračuje stanovenou mez (nastaveno na $160\,\mu s$, čemuž odpovídá přibližně $2.75\,cm$), měření je považováno za chybné. V opačném případě je měření započteno do průměru podle následujícího vzorce:

$$\mathrm{avg}_{\mathrm{new}} = \frac{6 \cdot \mathrm{avg}_{\mathrm{old}} + t_1 + t_2}{8},$$

kde $\mathrm{avg}_{\mathrm{old}}$ je předchozí hodnota proměnné `sonic.avg`, $\mathrm{avg}_{\mathrm{new}}$ je nová hodnota proměnné `sonic.avg`, $t_1$ je doba měření prvního ultrazvukového senzoru HC-SR04 a $t_2$ je doba měření druhého senzoru HC-SR04.

Efektivně se tedy nový průměr vypočte jako vážený průměr předchozího průměru s váhou 0.75 a nově naměřené vzdálenosti s váhou 0.25. Díky tomu, že v proměnné `sonic.avg`

zůstávají zohledněny předchozí měření, se vyhladí náhodné nárazové chyby v měření. I přes to, že je hodnota měření započtena, aby měření bylo považováno za úspěšné, musí navíc splňovat podmínku, že odchylka nově naměřené vzdálenosti ($\frac{t_1+t_2}{2}$) od průměru ($\text{avg}_{\text{new}}$) nesmí přesáhnout stanovenou mez (nastaveno na $80\,\mu s$ - přibližně $1.38\,cm$). V případě, že dojde k neúspěšnému měření, je zvýšen čítač `sonic.fail`. V případě, že je měření úspěšné, čítač `sonic.fail` je snížen. Na hodnotu `sonic.fail` reaguje úloha `watertank_control` (viz kapitola 8.6.5).

### 8.6.5 Řízení nádrže (úloha watertank_control)

Řízení nádrže spočívá v dopouštění a ohřevu vody. Přičemž ohřev a dopouštění je dovoleno pouze v případě, že nádrž není zamčena (počítač č.1 z nádrže nevypouští vodu). Zda je nádrž potřeba dopustit, se zjišťuje na základě naměřené výšky hladiny (`sonic.avg`). Zda je vodu v nádrži potřeba ohřát, se zjišťuje na základě naměřené teploty (`temperature.temp`) a minimální teploty vyžádané počítačem č.1 (`watertank.lock_temp_min`). Ohřev vody je dovolen, pouze pokud je nádrž plně napuštěna, aby bylo zajištěno, že jsou ohřívače během ohřívání vždy plně pod vodou.

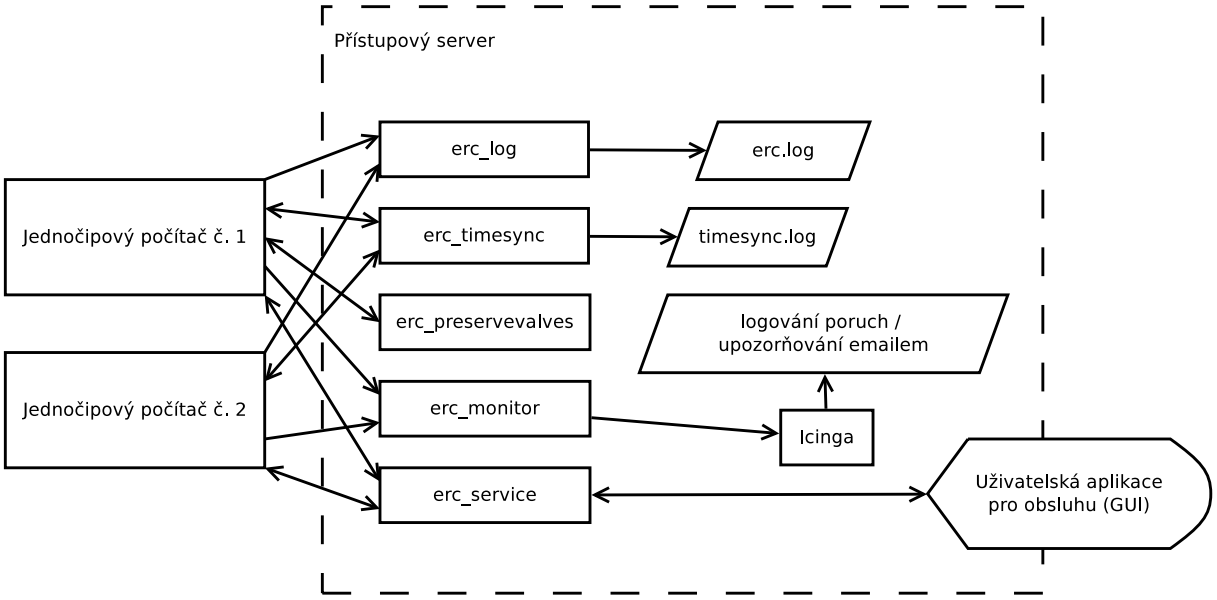
Jak pro dopouštění, tak pro ohřev se navíc používá hystereze, aby se předešlo nežádoucím přechodům mezi stavy kvůli mírným odchylkám v měření. Pro ohřev vody je hystereze nastavena na 0.5 °C. Pro napouštění nádrže je hystereze nastavena na $80\,\mu s$ (tedy přibližně $1.38\,cm$). Při ohřevu a napouštění se také kontroluje správnost měření (`temperature.ok` a `sonic.fail`). Pokud je měření chybné, ohřev, resp. napouštění, se zastaví.

Během napouštění se voda kontrolovaně míchá ze dvou vodních zdrojů (venkovní vrt a vodovod). Míchání je realizováno střídou na základě proměnné `watertank.filling_ticks`, která se inkrementuje, jen když probíhá napouštění. Již jsme dosáhli zjištění, že míchání 1:1 (časově) poskytuje stále vyhovující kvalitu vody. Momentálně snižujeme střídu ventilu od vodovodu ve prospěch využití vody z vrtu. Ventil pro napouštění z vrtu je tedy během napouštění stále otevřen, zatímco ventil od vodovodu se střídavě zapíná a vypíná dle nastavené střídy.

# Kapitola 9

# Implementace přístupového serveru

Na obrázku 9.1 je ilustrována komunikace jednotlivých částí systému uvnitř serveru a okolních zařízení a aplikací.




**Obrázek 9.1:** *Ilustrace komunikace uvnitř přístupového serveru a okolí*

Pro význam jednotlivých částí uvnitř přístupového serveru viz následující podkapitoly.

## 9.1 Běžné služby

Na přístupovém serveru běží některé běžně známé služby. Z těch, které se týkají tohoto vestavěného systému, jsou to:

- SSH server, pomocí kterého je možné server spravovat a přistupovat k ethernetové síti vestavěného systému pomocí ERC nástrojů (viz kapitola 9.4) a tunelovat spojení k soketu služby `erc_service` (viz kapitola 9.3.5).

- VPN, pomocí které je možné získat přístup přímo do ethernetové sítě vestavěného systému.
- NTP, pomocí kterého je synchronizován čas, který je dál synchronizován s jednočipovými počítači (viz kapitola 9.3.2).
- Icinga, pomocí které jsou (mimo jiné) monitorovány služby na přístupovém serveru. Icinga také přijímá výstup monitorovací služby `erc_monitor` (viz kapitola 9.3.4). Při detekování poruchy či jiného nežádoucího stavu Icinga o této skutečnosti reportuje emailem.

## 9.2 Komunikační knihovna ERC

Pro komunikaci s jednočipovými počítači jsem implementoval jednoduchou knihovnu `erc` (ethernet remote control). Knihovna je implementována v jazyce Python 3 v souboru `erc.py` a je možné ji importovat pomocí příkazu '`import erc`'. Tato knihovna zajišťuje vytvoření raw soketu (AF_PACKET, SOCK_RAW) pro komunikaci po ethernetovém rozhraní. Knihovna umožňuje přijímání a odesílání paketů kompatibilních s jednočipovými počítači včetně kontroly kontrolního součtu CRC-16-CCITT při přijímání a výpočtu a přidání kontrolního součtu při odesílání (viz kapitola 9.2.1).

Vytvoření soketu rodiny AF_PACKET vyžaduje oprávnění superuživatele. To znamená, že aplikace využívající knihovnu `erc` je nutné spouštět za uživatele root. Běh aplikací s oprávněními uživatele root přináší bezpečnostní rizika a je obecně považováno za nevhodné pro případy, jako je tento. Aplikace by proto po vytvoření soketu měla své oprávnění snížit, a to nejpozději před přijetím jakékoliv komunikace. Pro tyto účely jsem přidal do knihovny `erc` také funkci `yield_rights`, která změní oprávnění aplikace na uživatele `erc`.

### 9.2.1 API

Knihovna `erc` poskytuje třídu `ERC` a funkci `yield_rights`.

- Funkce `yield_rights` změní uživatele aplikace na `erc`, čímž zahodí oprávnění superuživatele.
- Třída `ERC` při instanciaci očekává jako vstup EtherType (tedy `0x0A01` pro komunikaci s bootloaderem, nebo `0x0A02` pro komunikaci s aplikací). Objekt třídy `ERC` poskytuje tyto metody:
  - `send` - Očekává parametry: cílová mac, data. Funkce `send` doplní potřebný počet nul na správnou délku paketu a spočítá a přidá kontrolní součet.
  - `recv` - Čeká na přijetí paketu. Vrací tuple: cílová mac, zdrojová mac, data. Funkce `recv` zkontroluje, zda má přijatý paket odpovídající EtherType a zkontroluje a odebere kontrolní součet. Pakety, které neprojdou některou z kontrol, jsou zahazovány. Volitelně je možné předat parametr `timeout` typu float v sekundách. Je-li předán tento parametr, funkce `recv` vrátí None, pokud ve zvoleném čase nedorazí žádný vyhovující paket.

**Příklad použití**

```
import erc;

dst = bytes.fromhex("02AABBCCDDEE");
e = erc.ERC(0x0A01);
e.send(dst, b'CB');
```

Zašle paket typu 'CB' (viz tabulka 7.2) s EtherType 0x0A01 na mac adresu 02:AA:BB:CC:DD:EE. Tedy rebootuje jednočipový počítač s ethernetovou adresou 02:AA:BB:CC:DD:EE (za předpokladu, že na něm běží bootloader z kapitoly 7).

## 9.3 Služby postavené na knihovně ERC

### 9.3.1 erc_log

Služba erc_log neposkytuje žádnou funkci, bez které by vestavěný systém nemohl fungovat a nikdy neodesílá žádné pakety. Pouze zajišťuje logování paketů zejména pro účely ladění, případně také pro další analýzy logovaných dat (o analýze logovaných dat viz kapitola 11). Služba erc_log loguje tyto pakety: 'SP', 'SH', 'SW', 'SN', 'SS' (pro význam a formát těchto paketů viz tabulky 8.1, 8.4 a 8.6). Služba erc_log reaguje na signál SIGUSR1 tím, že uzavře a znovu otevře výstupní logovací soubor. To je užitečné v kombinaci s logrotate pro rotování logů.

**Úryvek logu**

```
   Wed Apr 11 17:52:29 2018 b'02414b560002' relays: b'0000' +b'0000' -b'0000'
= b'0000', flags: b'0000', PORTS: af 1f ff fb, PINS: f 1c ff ff
   Wed Apr 11 17:52:29 2018 b'02414b560002' water tank: sonic_fail = 0, level
= 124, filling = 0, filled = 0, heated = 1, tempok = 3, locked = 1, temp_min
= 22.0000 °C, timeout = 2865s
   Wed Apr 11 17:52:29 2018 b'02414b560001' relays: b'0101' +b'0000' -b'0000'
= b'0101', flags: b'0700', PORTS: ff 1e fe fb, PINS: ff 1e fe 83
   Wed Apr 11 17:52:30 2018 b'02414b560001' water renewal: active = 1, valve
= 0, req_lock = 0, lock = 3, VALVES: 363, 5835, 5835
   Wed Apr 11 17:52:31 2018 b'02414b560002' temperature sensor: id = 0, status
= OK, temperature = 22.5000 °C
   Wed Apr 11 17:52:33 2018 b'02414b560002' sonic: val1 = 128, val2 = 131, fail
= 0, val = 125
```

Z řádků relays, flags, PORTS a PINS je poznat, že ani jeden z jednočipových počítačů nemá maskovány žádné relé. Počítač č.2 má všechny relé vypnuty. Počítač č.1 má zapnuto relé pro hlavní osvětlení a ventil č.1. Dle příznaků jsou na počítači č.1 povoleny ventily č.1, č.2 a č.3, osvětlení není manuálně zapnuto.

Z řádku water tank je poznat, že je nádrž uzamčena, v nádrži chybí přibližně 17 *cm* vody, minimální požadovaná teplota vody v nádrži je 22 °C a za přibližně tři čtvrtě hodiny bude nádrž sama uvolněna, pokud do té doby jednočipový počítač č.2 neobdrží žádný paket 'CL' nebo 'CU'.

Z řádku `water renewal` je poznat, že počítač č.1 obměňuje vodu ventilem 0 (č.1), zbývá vypouštět přibližně 80 $s$ ventilem č.1 a přibližně 22 minut ventily č.2 a č.3. Zámek vyprší za 3 iterace (48 $s$), takže do uvolnění nádrže počítač č.1 nestihne doobměnit ani ventil č.1.

Z řádku `temperature sensor` je poznat, že měření teploty proběhlo úspěšně a voda v nádrži má teplotu 22.5 °C.

Na řádku `sonic` je vidět, že měření výšky hladiny proběhlo úspěšně, jaké jsou konkrétní naměřené hodnoty z obou senzorů (rozdíl mezi senzory je 30 $\mu s$, čemuž odpovídá vzdálenost 5.1 $mm$) a že nový průměrný úbytek hladiny je 125, což oproti hodnotě 124 z předchozího řádku `water tank` představuje pokles hladiny o 1.7 $mm$. Vzhledem k tomu, že počítač č.1 vypouští, není překvapivé, že počítač č.2 zaznamenává pokles hladiny.

### 9.3.2 erc_timesync

Služba `erc_timesync` zajišťuje synchronizaci času na jednočipových počítačích. Funguje tak, že čeká na pakety typu '`ST`' a při přijetí paketu typu '`ST`' porovná čas jednočipového počítače s lokálním časem a reaguje paketem typu '`SF`', nebo '`SS`' podle toho, jak velký je časový rozdíl. V případě, že se odlišuje časová zóna, pošle také paket typu '`CZ`' pro nastavení správné časové zóny.

Služba `erc_timesync` také loguje informace o synchronizaci času do souboru pro účely ladění. Logovány jsou i pakety typu '`AF`', '`AS`' a '`AZ`'. Pro význam a formát paketů '`ST`', '`SF`', '`SS`', '`AF`', '`AS`' a '`AZ`' viz tabulka 8.1. Pro podrobnější popis, jak funguje synchronizace času, viz kapitola 8.2.1.

Služba `erc_timesync` reaguje na signál SIGUSR1 tím, že uzavře a znovu otevře výstupní logovací soubor. To je užitečné v kombinaci s logrotate pro rotování logů.

#### Úryvek logu

```
Wed Apr 11 21:30:53 2018: b'02414b560001': ST: 390009613804 -
    390009613805 = -1 (+7200 x +7200)
Wed Apr 11 21:30:53 2018: b'02414b560001':   setting speed to 7
    a0f
Wed Apr 11 21:30:53 2018: b'02414b560001': AS: speed set to 7a0f
Wed Apr 11 21:30:59 2018: b'02414b560002': ST: 2048 -
    390009615108 = -390009613060 (+0 x +7200)
Wed Apr 11 21:30:59 2018: b'02414b560002':   time zone mismatch -
     updating
Wed Apr 11 21:30:59 2018: b'02414b560002':   offset is exceeding
    100 - forcing local time
Wed Apr 11 21:30:59 2018: b'02414b560002': AZ: time zone set to
    7200
Wed Apr 11 21:30:59 2018: b'02414b560002': AF: time forced to
    390009615108
Wed Apr 11 21:31:01 2018: b'02414b560001': ST: 390009615852 -
    390009615853 = -1 (+7200 x +7200)
Wed Apr 11 21:31:01 2018: b'02414b560001':   setting speed to 7
    a0f
Wed Apr 11 21:31:01 2018: b'02414b560001': AS: speed set to 7a0f
```

```
Wed Apr 11 21:31:07 2018: b'02414b560002': ST: 390009617156 -
    390009617156 = 0 (+7200 x +7200)
Wed Apr 11 21:31:07 2018: b'02414b560002':   setting speed to 7
    a10
Wed Apr 11 21:31:07 2018: b'02414b560002': AS: speed set to 7a10
```

Z logu je poznat, že jednočipový počítač č.1 je momentálně o $\frac{1}{256}\,s$ opožděn, a proto je mu nastavována perioda časovače na `0x7A0F`. Jednočipový počítač č.2 má nejdřív čas $8\,s$ (právě nabootoval), proto je mu vnucen absolutní lokální čas. Také jeho časová zóna ještě není nastavena (+0), a tak je aktualizována na SELČ (+7200). Při příštím paketu '`ST`' od počítače č.2 je již čas včetně časové zóny synchronizován, a proto je již pouze nastavena perioda časovače na výchozí hodnotu `0x7A10`.

### 9.3.3 erc_preservevalves

Služba `erc_preservevalves` zajišťuje zachování akumulovaných hodnot ventilů po resetu jednočipového počítače č.1. Funguje tak, že služba přijímá pakety typu '`SN`' a když takový paket přijme, zkontroluje, zda nejsou akumulované hodnoty ventilů příliš nízké, což by indikovalo stav po resetu. Pokud ne, uloží si hodnoty ventilů a aktuální čas. Pokud služba detekuje, že došlo k resetu, použije poslední známé akumulované hodnoty ventilů a přičte k nim, o kolik by se hodnoty ještě navíc zvýšili od času posledního známého stavu. Výsledné součty pak služba pošle jednočipovému počítači č.1 pomocí paketu '`CV`'. Pro formát a význam paketů '`SN`' a '`CV`' viz tabulka 8.4.

Díky službě `erc_preservevalves` reboot jednočipového počítače č.1 (např. v důsledku aktualizace firmwaru nebo výpadku napájení) nezpůsobí vynulování akumulovaných stavů ventilů, a tedy i přes rebooty by mělo být obměněno správné množství vody, aniž by bylo nutné stavy ventilů manuálně nastavovat.

### 9.3.4 erc_monitor

Služba `erc_monitor` zajišťuje monitorování vestavěného systému a detekci poruch či jiných nežádoucích stavů. Výsledky monitorování jsou předávány službě Icinga, která běží taktéž na přístupovém serveru (viz kapitola 9.1).

Služba `erc_monitor` dokáže detekovat problémy typu:

- úplný výpadek jednočipového počítače - kritická chyba
- náhlá změna akumulovaných hodnot ventilů (např. když dojde k resetu jednočipového počítače č.1) - varování
- příliš vysoké akumulované hodnoty ventilů, kvůli kterým bude nutné obměňovat vodu i přes noc - varování
- probíhající obměna vody bez uzamčené nádrže - kritická chyba

### 9.3.5 erc_service

Služba `erc_service` zpřístupňuje informace a ovládání vestavěného systému prostřednictvím jednoduchého API postaveného na formátu JSON. Službu `erc_service` využívá uživatelská aplikace pro obsluhu (viz kapitola 10).

Služba `erc_service` naslouchá na unixovém soketu, pomocí kterého ji mohou klienti kontaktovat a vyžádat si informace o stavu vestavěného systému, případně zaslat příkaz na změnu stavu. Tabulka 9.1 znázorňuje zprávy, které služba `erc_service` přijímá a odesílá.

Unixový soket má nastavena práva tak, že se k němu mohou připojit pouze uživatelé skupiny `erc`. Tím je řešeno zabezpečení, aby službu `erc_service` mohli používat pouze oprávnění uživatelé.

Služba `erc_service` reaguje také na pakety od jednočipových počítačů a při přijetí paketu informujícím o stavu jednočipového počítače si aktualizuje svůj udržovaný stav a pokud se nový stav liší od předchozího, rozešle nový stav klientům, kteří se registrovali k informování o změnách stavu pomocí zprávy `[''REG FOR STATE UPDATE'']`.

| Zprávy přijímané službou `erc_service` | |
|---|---|
| Zpráva | Popis |
| `[''GET STATE'']` | Vyžádá si aktuální stav vestavěného systému. |
| `[''REG FOR STATE UPDATE'']` | Registruje se k informování aktualizacích stavu systému. |
| `[''CTRL'', PC, ID, CLASS, NAME, DATA]` | Požadavek na změnu stavu.<br>Za ID je potřeba dosadit identifikaci (pořadí) jednočipového počítače.<br>Za ID je potřeba dosadit identifikaci (pořadí) konkrétního ovladače, kterému má být příkaz zaslán.<br>Za CLASS je potřeba dosadit třídu ovladače (slouží pouze pro kontrolu).<br>Za NAME je potřeba dosadit název příkazu.<br>Za DATA je potřeba dosadit data, která mají být ovladači s příkazem předána. |
| Zprávy odesílané službou `erc_service` | |
| Zpráva | Popis |
| `[''STATE'', STAV]` | `STAV` je předaná datová struktura, která nese informaci o stavu všech ovladačů.<br>Tato zpráva může být zaslána buď v reakci na zprávu `[''GET STATE'']`, nebo automaticky při změně stavu, pokud se k tomu klient registroval zprávou `[''REG FOR STATE UPDATE'']`. |
| `[''REGISTERED'']` | Odpověď na `[''REG FOR STATE UPDATE'']` |
| `[''CTRL OK'']` | Kladná odpověď na zprávu `[''CTRL'', ...]` |
| `[''ERROR'', CHYBA]` | Informace o chybě. Parametr `CHYBA` upřesňuje k jaké chybě došlo. |

**Tabulka 9.1:** *Zprávy podporované službou `erc_service`*

### Ovladače

V tabulce 9.2 jsou vyjmenovány třídy ovladačů podporované službou `erc_service`.

| Třída ovladače | Atributy | Přijímané ovládací zprávy [name, data] | Popis |
|---|---|---|---|
| Relay | masked (boolean) - skutečný stav relé<br>auto (boolean) - automatický stav<br>mask_on (boolean) - manuálně zapnuto<br>mask_off (boolean) - manuálně vypnuto | ‘‘setmask’’, enum {‘‘Off’’, ‘‘On’’, ‘‘Auto’’} | Stav a ovládání relé |
| Flag | boolean - stav příznaku | ‘‘set’’, boolean | Stav a ovládání příznaku |
| Temperature | float - teplota v °C | - | Stav teplotního senzoru |
| Sonic | sensor1 (float) - vzdálenost v metrech<br>sensor2 (float) - vzdálenost v metrech<br>average (float) - vzdálenost v metrech<br>fail (boolean) | - | Stav senzorů vzdálenosti HC-SR04 |
| WaterTank | level (float) - naplnění nádrže v rozsahu 0.0 až 1.0<br>filling (boolean) - napouštění,<br>heated (boolean) - ohřáto,<br>locked (boolean) - uzamčeno | - | Stav nádrže |
| WaterRenewal | active (boolean) - probíhá vypouštění<br>valves (pole struktur) {<br>renew (float) - zbývá obměnit v sekundách<br>active (boolean) - právě obměňováno<br>neglected (enum {’’No’’, ‘‘Low’’, ‘‘High’’})<br>} | ‘‘adjust’’, pole floatů | Stav obměny vody včetně akumulovaných hodnot všech ventilů s možností přičítat hodnotu k akumulovaným hodnotám |

**Tabulka 9.2:** *Ovladače podporované službou* `erc_service`

## 9.4 Nástroje ERC

Nástroje vyjmenované v této kapitole za běžných okolností neběží, ani nejsou automaticky spouštěny. Slouží pro vývoj vestavěného systému, ladění a manuální zásahy do systému.

### 9.4.1 erc_cmd

Nástroj `erc_cmd` usnadňuje posílání obecně libovolných příkazů. Aplikace `erc_cmd` očekává jako argumenty: cílovou mac adresu, EtherType a data paketu. Data paketu mohou být předána ve více argumentech, které jsou zkonkatenovány. Pokud argument začíná číslicí nebo některým ze znaků plus, mínus, dvojtečka, je považován za číselný argument kódovaný jako big-endian. V opačném případě, je argument považován za řetězcový.

Řetězcový argument je použit bez jakýchkoliv úprav přímo v předané podobě. Číselný argument je očekáván ve formátu `velikost:hodnota`, kde `velikost` udává velikost číselné hodnoty v bajtech a `hodnota` udává hodnotu čísla. Údaj `velikost`, případně i včetně dvojtečky, může být vynechán. V takovém případě je velikost určena implicitně. Jedná-li se o číslo zadané v binárním nebo hexadecimálním formátu, je velikost odvozena z řetězcové délky parametru (například argument 0x000001 znamená velikost 3 bajty a hodnota 1). V ostatních případech je velikost implicitně 1 bajt (například argument 300 způsobí selhání, protože se hodnota 300 nevejde do jednoho bajtu).

#### Příklady použití

```
# ./erc_cmd 02:AA:BB:CC:DD:EE 0x0A01 CB;
```

Zašle paket s EtherType `0x0A01` typu '`CB`' (viz tabulka 7.2) na mac adresu `02:AA:BB:CC:DD:EE`. Tedy rebootuje jednočipový počítač s ethernetovou adresou `02:AA:BB:CC:DD:EE` (za předpokladu, že na něm běží bootloader z kapitoly 7).

```
# ./erc_cmd 02:AA:BB:CC:DD:EE 0x0A02 CG 0x0001 0x0001;
```

Zašle paket s EtherType `0x0A02` typu '`CG`' (viz tabulka 8.4) s parametry flags=0x0001, mask=0x0001 na mac adresu `02:AA:BB:CC:DD:EE`. Tedy nastaví příznak 1 (`FLAG_LIGHT`) a tím rozsvítí světla v odchovně (za předpokladu, že se na této mac adrese nachází jednočipový počítač č.1 z kapitoly 8.5).

```
# ./erc_cmd 02:AA:BB:CC:DD:EE 0x0A02 CV 2:5 2:10 2:-5;
```

Zašle paket s EtherType `0x0A02` typu '`CV`' (viz tabulka 8.4) s parametry ventilů 5 10 -5 na mac adresu `02:AA:BB:CC:DD:EE`. Tedy zvýší akumulovaný stav ventilu č. 1 o 5 iterací, ventilu č. 2 o 10 iterací a stav ventilu č. 3 sníží o 5 iterací.

### 9.4.2 erc_cq

Nástroj `erc_cq` usnadňuje přepnutí jednočipového počítače do flashovacího režimu. Aplikace `erc_cq` očekává jako argument mac adresu jednočipového počítače, který má být přepnut do flashovacího režimu. Nástroj funguje tak, že čeká na pakety od zadané mac adresy a na některé reaguje. Pokud nástroj obdrží paket typu '`SB`' (jednočipový počítač bootuje), odpoví na něj paketem typu '`CQ`' (přerušení bootování a přepnutí do flashovacího režimu). Pokud nástroj obdrží paket typu '`AQ`' (potvrzení o přepnutí do flashovacího režimu), nástroj skončí s úspěchem. Pro formát paketů '`SB`', '`CQ`' a '`AQ`' viz tabulka 7.2.

Volitelně lze přidat další argument `reboot`, který způsobí, že se ihned po spuštění `erc_cq` odešle příkaz '`CB`', který jednočipový počítač rebootuje.

**Příklad použití**

```
# ./erc_cq 02:AA:BB:CC:DD:EE reboot;
sending reboot command ('CB')
reboot confirmation ('AB') received
boot notification received ('SB'), sending quit to setup ('CQ')
flash mode entered ('AQ' received)
#
```

### 9.4.3 erc_prog

Nástroj `erc_prog` slouží k programování jednočipového počítače, na kterém běží bootloader z kapitoly 7. Aplikace `erc_prog` očekává 3 až 4 argumenty. První 3 argumenty jsou: mac adresa cílového jednočipového počítače, binární soubor s programem a klíčové slovo `check`, nebo `flash`.

V případě, že je použito klíčové slovo `check`, nástroj pouze zkontroluje (pomocí paketů typu '`QR`'), že ve flash paměti zadaného jednočipového počítače je nahrán program z referenčního souboru. Pokud je použito klíčové slovo `flash`, nástroj `erc_prog` nahraje (pomocí paketů '`QR`' a '`CW`') do flash paměti jednočipového počítače program ze zadaného referenčního souboru. Pakety typu '`QR`' jsou při flashování použity pro zjištění, zda se daná stránka flash paměti již neshoduje s referenčním souborem. Pokud se stránka již shoduje, není přehrána znovu, aby se zbytečně neopotřebovávala flash paměť. Pro formát paketů typu '`QR`' a '`CW`' viz tabulka 7.2.

Režim `check` je možné provést i za běhu aplikačního programu. Aby fungoval režim `flash`, musí být jednočipový počítač přepnut do flashovacího režimu. Pokud je spuštěn nástroj `erc_prog` v režimu `flash` a jednočipový počítač není ve flashovacím režimu, tak flashování selže při první stránce, kterou bude potřeba změnit.

Poslední argument je volitelný offset ve flash paměti jednočipového počítače. Pomocí něj je možné kontrolovat a flashovat flash paměť jednočipového počítače i v jiných oblastech než na začátku. To je užitečné například pro kontrolu bootloaderu (viz příklad kontroly bootloaderu níže). Bootloader není možné přes ethernet změnit díky nastavenému lock bitu `BLB11 = 0` (viz kapitola 7.1).

**Příklad použití**

```
# ./erc_prog 02:AA:BB:CC:DD:EE proj.bin check
checking page 0 of 4 ... match
checking page 1 of 4 ... match
checking page 2 of 4 ... differs
checking page 3 of 4 ... match
program does not match
# ./erc_prog 02:AA:BB:CC:DD:EE proj.bin flash
checking page 0 of 4 ... match
checking page 1 of 4 ... match
checking page 2 of 4 ... writing ... failed
failed to write page, perhaps remote is not in flash mode
# ./erc_cq 02:AA:BB:CC:DD:EE reboot
sending reboot command ('CB')
reboot confirmation ('AB') received
```

```
boot notification received ('SB'), sending quit to setup ('CQ')
flash mode entered ('AQ' received)
# ./erc_prog 02:AA:BB:CC:DD:EE proj.bin flash
checking page 0 of 4 ... match
checking page 1 of 4 ... match
checking page 2 of 4 ... writing ... done
checking page 3 of 4 ... match
program updated
# ./erc_prog 02:AA:BB:CC:DD:EE proj.bin check
checking page 0 of 4 ... match
checking page 1 of 4 ... match
checking page 2 of 4 ... match
checking page 3 of 4 ... match
program match
# ./erc_cmd 02:AA:BB:CC:DD:EE 0x0A01 CB
#
```

V tomto příkladu flashuji malý (4 stránkový) program `proj.bin` do jednočipového počítače s mac adresou `02:AA:BB:CC:DD:EE`. Nejdříve pomocí `check` porovnám referenční soubor a flash paměť jednočipového počítače. Z výstupu nástroje `erc_prog` je poznat, že pouze stránka s indexem 2 se liší od referenčního souboru. Následně se pokouším pomocí režimu `flash` program aktualizovat, což selže, protože jednočipový počítač není ve flashovacím režimu. Pak pomocí nástroje `erc_cq` jednočipový počítač rebootuji do flashovacího režimu. Další pokus o aktualizaci programu je již úspěšný. Po aktualizaci programu znovu porovnám referenční soubor s flash pamětí. Tentokrát již program ve flash paměti odpovídá referenčnímu souboru. Nakonec jednočipový počítač rebootuji do nového programu.

Binární soubor (.bin) je možné získat z ELF souboru například pomocí objcopy:

```
$ avr-objcopy -j .text -j .data -O binary proj proj.bin
```

**Příklad kontroly bootloaderu**

```
# ./erc_prog 02:AA:BB:CC:DD:EE bootloader.bin check 224
checking page 0 of 25 ... match
checking page 1 of 25 ... match
...
checking page 24 of 25 ... match
program match
#
```

V tomto příkladu kontroluji bootloader. Řádky kontrolující stránky 2 až 23 jsem vypustil. Z výstupu nástroje `erc_prog` je poznat, že bootloader ve flash paměti odpovídá referenčnímu souboru `bootloader.bin`. Stránka bootloaderu se vypočte tak, že se bajtová adresa bootloaderu (`0x7000`) vydělí velikostí stránky (128): $\frac{0x7000}{128} = 224$.

### 9.4.4 erc_dumpmem

Nástroj `erc_dumpmem` může pomoct zejména při ladění. Pomocí něj je možno získat z jednočipového počítače obraz celého paměťového prostoru. Funguje tak, že pošle jednočipovému

počítači požadavek 'QD' a složí dohromady data z odpovědí 'RD'. Výsledek vypíše na standardní výstup. Získaná data je pak možné například nahrát do gdb a následně nad nimi provést další analýzu. Pro význam a formát paketů 'QD' a 'RD' viz tabulka 7.2.

**Příklad použití v kombinaci s gdb**

```
# ./erc_dumpmem 02:41:4B:56:00:01 > /tmp/dumpmem
page 0 of 3 received
page 1 of 3 received
page 2 of 3 received
#
```

```
$ avr-gdb ./dev1
...
(gdb) target sim
Connected to the simulator.
(gdb) load
...
(gdb) start
...
(gdb) restore /tmp/dumpmem binary 0x800000
Restoring binary file /tmp/dumpmem into memory (0x800000 to 0x800860)
(gdb) start
...
(gdb) next
...
(gdb) info locals
pins = 2080432384
time_sync = {time = 390134790852, last_sync = 0, tz_offset = 7200, sync = true}
light_ctrls = {control = true, state = 310}
fan_ctrls = {ticks = 127 '\177'}
flags = 1792
relay = {app = 1, mask_off = 0, mask_on = 0, masked = 1}
water_renewal = {active = false, active_valve = 0 '\000', requesting_lock = true, lock = 0, valve = {11177, 10933, 10933}}
tasks = {{func = 0x92 <task_time_advance>, arg = 0x800849, mask = 0, cmp = 0}, {func = 0xc0 <task_time_sync_req>, arg = 0x800847, mask = 2044, cmp = 0}, {func = 0x3fa <task_status_reports>,
 arg = 0x800828, mask = 2044, cmp = 16}, {func = 0x80a <task_switch_handler>, arg = 0x800822, mask = 252, cmp = 32}, {func = 0x8f6 <task_light_control>, arg = 0x800812, mask = 252,
 cmp = 48}, {func = 0xaec <task_fan_control>, arg = 0x80080a, mask = 4092, cmp = 48}, {func = 0xc0a <task_water_renewal>, arg = 0x800802, mask = 4092, cmp = 64}, {
 func = 0x6bc <task_setrelays>, arg = 0x800845, mask = 252, cmp = 64}, {func = 0xb90 <task_water_renewal_report>, arg = 0x800843, mask = 4092, cmp = 80}, {func = 0x0 <__vectors>,
 arg = 0x0 <__vectors>, mask = 0, cmp = 0}}
cmds = {{op = "CB", func = 0x618 <cmd_reboot>, arg = 0x800841}, {op = "CF", func = 0x1ea <cmd_time_force>, arg = 0x80083f}, {op = "CS", func = 0x392 <cmd_time_speed>, arg = 0x80083d}, {
 op = "CZ", func = 0x3ca <cmd_time_set_tz>, arg = 0x80083b}, {op = "CM", func = 0x71c <cmd_relay_mask>, arg = 0x800839}, {op = "CG",
```

```
            func = 0x652 <cmd_bitmask16_update>, arg = 0x800837}, {
30      op = "AL", func = 0xebc <ack_watertank_lock>, arg = 0x800835}, {op =
            "CV", func = 0xf10 <cmd_watertank_valve_adjust>, arg = 0x800833
          }, {op = "\000", func = 0x0 <__vectors>,
31      arg = 0x0 <__vectors>}}
32  (gdb) p/x OCR1A
33  $1 = 0x7a0f
34  (gdb) p/x {DDRA, DDRB, DDRC, DDRD}
35  $2 = {0x1, 0xbf, 0xff, 0x83}
36  (gdb)
```

Pomocí gdb načtu zkompilovaný program pro jednočipový počítač č.1. Pak spustím program pomocí integrovaného simulátoru (příkazy target sim, load, start). A pak pomocí restore nahraji obraz paměti získaný pomocí `erc_dumpmem`. Nahrání paměti způsobí také to, že se registry dostanou do nekonzistentního stavu, proto znovu spustím simulaci od začátku pomocí start (tentokrát už s nahraným obrazem paměti) a provedu jednou next, aby se posunul registr $sp na správnou hodnotu, pokud je to potřeba. Následně už je možné pracovat s lokálními proměnnými funkce main.

Ve výstupu příkazu info locals je možné vidět vnitřní stav proměnných vytvořených ve funkci main. Je například vidět, že jsou rozsvícená světla (`relay.masked` = 1), aktuální časová zóna je SELČ (`time_sync.tz_offset` = 7200), hodnota `light_ctrls.state` je na konci (viz obrázek 8.1), stav obměny vody (`water_renewal`) a také je vidět, že proměnné pro plánovač (`tasks` a `cmds`) jsou nastaveny správně.

Protože `erc_dumpmem` krom paměti SRAM získá také obraz registrů, je možné vypisovat i stavy registrů jednočipového počítače, jaké byly při získávání obrazu paměťového prostoru (řádky p/x). Je však třeba brát v potaz, že nejde o snímek, a různé registry byly čteny v různý čas. Registry, které se mohou samy od sebe měnit (například vstupy, hodnoty časovačů), mohou být vzájemně v nekonzistentním stavu - tedy ve stavu, který ve skutečnosti nikdy v žádném čase nenastal.

# Kapitola 10

# Implementace uživatelské aplikace pro obsluhu

Uživatelská aplikace pro obsluhu slouží k manuálnímu ovládání vestavěného systému. Pomocí této aplikace je možné zjišťovat stav vestavěného systému a ovládat jeho funkce jako například osvětlení, obměnu vody, jednotlivá relé.

Aplikace se pomocí unixového soketu připojuje ke službě `erc_service` (viz kapitola 9.3.5).

Aplikace je implementována v jazyce Python3 s využitím knihovny GTK+3 pro grafické rozhraní a knihovny gettext pro lokalizaci textu. Díky knihovně GTK+3 je aplikace dobře přenositelná. Krom běžného desktopu je možné ji spustit také například na smartphonu se systémem Android a díky tomu, že je knihovna GTK+3 implementována i s ohledem na zařízení s dotykovou obrazovkou, se s aplikací dá poměrně dobře pracovat i na těchto zařízeních. V případě potřeby je možné díky backendu broadway vytvořit webový server s touto aplikací a aplikaci ovládat přes webový prohlížeč.

Na obrázku 10.1 je snímek uživatelské aplikace běžící ve webovém prohlížeči.


**Obrázek 10.1:** *Snímek obrazovky grafické uživatelské aplikace pro obsluhu běžící ve webovém prohlížeči*

Uživatelská aplikace zobrazuje jednotlivé ovladače zpřístupněné službou `erc_service` (viz tabulka 9.2).

Třítlačítkové ovladače (Zap, Auto, Vyp) jsou ovladače typu `Relay`. Barva nadpisu ovladače `Relay` znázorňuje atribut `masked` - tedy reálný stav sepnutí relé. Tři tlačítka pod nadpisem fungují zároveň pro ovládání a zároveň pro znázornění stavu. Tyto tři tlačítka při stisknutí vyvolají pro daný ovladač zprávu typu `''setmask''` s parametrem `''Off''`, `''On''`, `''Auto''` podle toho, které konkrétní tlačítko bylo stisknuto. Tlačítko Zap se rozsvítí zeleně, pokud je atribut `mask_on` roven True. Tlačítko Vyp se rozsvítí červeně, pokud je atribut `mask_off` roven True. Tlačítko Auto svítí červeně nebo zeleně podle hodnoty atributu `auto`.

Dvoutlačítkové ovladače (Zap, Vyp) jsou ovladače typu `Flag`. Barva nadpisu znázorňuje stav příznaku. Tlačítka Zap a Vyp vyvolávají zprávu typu `''set''` s parametrem True nebo False podle toho, které tlačítko bylo stisknuto.

Ovladač s nadpisem „Obměna vody“ je ovladač typu `WaterRenewal`. Barva nadpisu znázorňuje stav atributu `active`. Hodnoty ventilů jsou vyjádřeny ve formě času, po který je potřeba ventilem vypouštět, aby se akumulovaná hodnota ventilu vynulovala. Konkrétní ventil zčervená, pokud jeho hodnota přesahuje práh, při kterém už se počítač č. 1 pokouší o obměnu vody. Zeleně je vyznačen aktivní ventil (kterým je právě vypouštěno). Tučně je vyznačen ventil s vysokou akumulovanou hodnotou, při které už bude prováděna obměna vody i během noci.

Ovladač s nadpisem „Teplota v nádrži“ je ovladač typu `Temperature` a zobrazuje aktuální měřenou teplotu.

Ovladač s nadpisem „Sonic“ je ovladač typu `Sonic`. Zobrazuje atributy `sensor1`, `sensor2` a `average` v tomto pořadí. Při nastaveném atributu `fail` se ovladač zobrazí tučně červeně.

Ovladač s nadpisem „Nádrž“ je ovladač typu `WaterTank`. Modrý obrazec znázorňuje plnost nádrže (atribut `level`). Nad tímto obrazcem je ikona znázorňující stav nádrže. Je-li aktivní atribut `locked`, je zobrazena ikona zámku. Pokud je atribut `filling` roven True a atribut `heated` roven False, je zobrazen zelený symbol znázorňující, že s nádrží není potřeba nic provádět. V ostatních případech je zobrazen symbol šipek znázorňující, že je nádrž připravována.

Na obrázku 10.2 je snímek uživatelské aplikace běžící v prostředí x11.


**Obrázek 10.2:** *Snímek obrazovky grafické uživatelské aplikace pro obsluhu běžící v prostředí X11*

Ze snímku je poznat, že je osvětlení momentálně zapnuto. Ventilátory běží, protože jsou manuálně zapnuty (jinak by neběžely). Povolený je pouze ventil č.1, ventily č.2 a č.3 mají

poměrně vysoké akumulované hodnoty. Ventil č.2 je dokonce vyznačen tučně, což znamená, že jeho akumulovaná hodnota přesáhla hranici, při které by byla voda obměňována i přes noc (kdyby byl ventil č.2 povolen). Voda v nádrži je právě ohřívána na požadovanou teplotu. Ventil pro dopouštění vody z vrtu a čerpadlo vody z vrtu jsou momentálně manuálně vypnuty (tlačítka „Vyp“ jsou červená). To nyní nemá vliv, protože se nedopouští voda do nádrže (tlačítka „Auto“ jsou také červená). Ale pokud by počítač č.2 dopouštěl nádrž, použil by k tomu pouze ventil pro dopouštění vodou z vodovodu. Dále je ze snímku poznat, že aktuální teplota nádrže je 19.25 °C a nádrž je napuštěna.

Na obrázku 10.3 je snímek uživatelské aplikace běžící na mobilním telefonu.


**Obrázek 10.3:** *Snímek obrazovky grafické uživatelské aplikace pro obsluhu běžící na smartphonu se systémem Android*

Aplikace se na obrazovku telefonu nevejde celá a je potřeba ji scrollovat. Na dotykové obrazovce je možné scrollovat tahem pomocí dvou prstů.

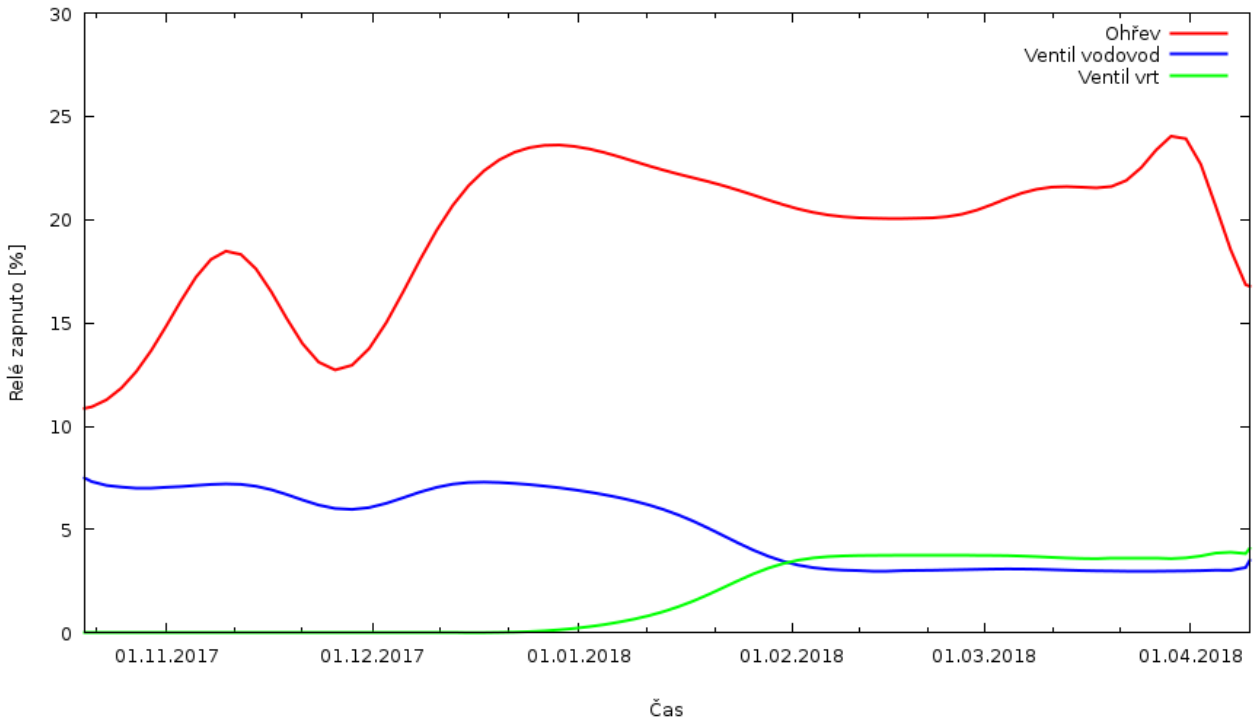
Na snímku je vidět, že je momentálně nádrž uzamčena, přestože nedochází k vypouštění. To je stav, který by za běžných okolností neměl nastat. Je to způsobeno tím, že počítač č.2 byl právě resetován a nádrž je inicializována do uzamčeného stavu (pro případ kdyby došlo k resetu počítače č.2 a počítač č.1 zrovna vypouštěl nádrž).

# Kapitola 11

# Analýza logovaných dat

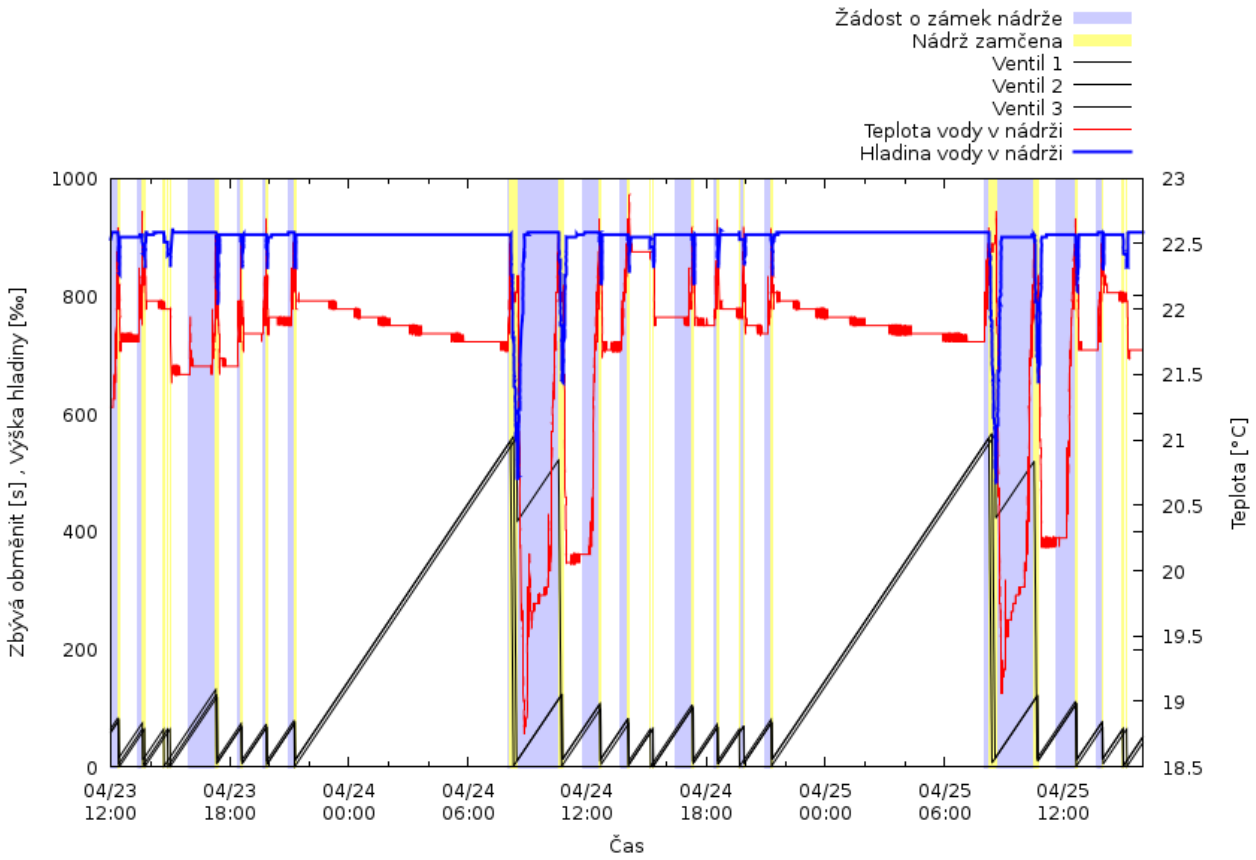
Z logovaných dat je možné získat některé zajímavé statistické informace.

Na obrázku 11.1 je vidět graf sepnutí relé v závislosti na čase pro jednočipový počítač č.2.




**Obrázek 11.1:** *Graf průměrného sepnutí relé v závislosti na čase na jednočipovém počítači č.2*

Z grafu je poznat, že v zimním období je potřeba více času věnovat ohřevu vody v nádrži. Začátkem ledna jsme začali využívat vodu z vrtu. Začátkem února jsme narazili na problémy s podezřením, že by mohly být způsobeny kvalitou vody, proto jsme raději přestali poměr využití vody z vrtu zvyšovat. Z toho, že se téměř nezměnil součet časů obou ventilů se dá také poznat, že průtok vody oběma ventily je přibližně stejný.



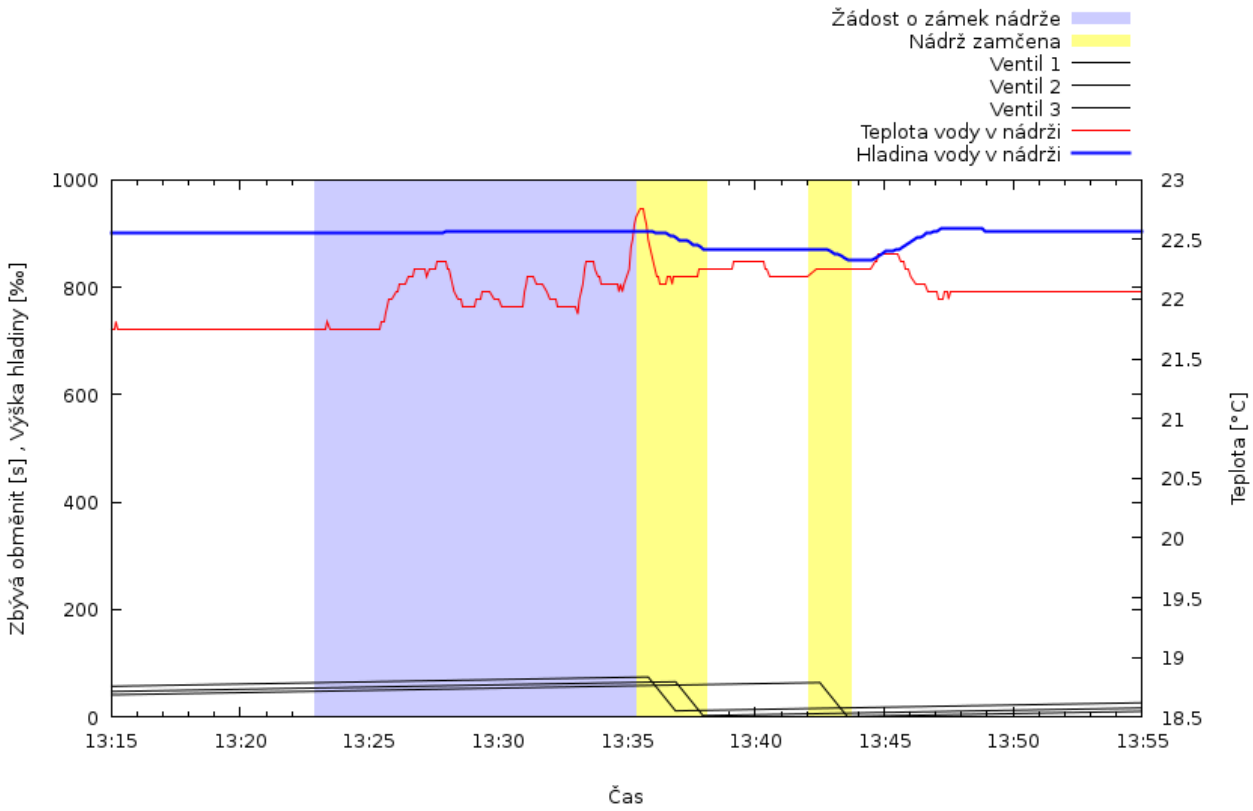

**Obrázek 11.2:** *Graf obměny vody v rámci dvou dnů*

Na obrázku 11.2 je vidět typický denní průběh obměny vody.

Černě jsou vyznačeny akumulované hodnoty ventilů ve formě času, po který je potřeba ventilem vypouštět. Modré oblasti vyznačují dobu, kdy počítač č.1 žádá počítač č.2 o uzamčení nádrže. Žluté oblasti vyznačují dobu, kdy je nádrž uzamčena. Výška hladiny nádrže (vyznačena modře) je vyjádřena v poměru plnosti nádrže dle měřeného rozsahu ($1000\,‰ = 0\,cm$ až $0\,‰ \doteq 41\,cm$ poklesu hladiny).

Z grafu je poznat, že během noci se voda neobměňuje a pouze se akumulují hodnoty ventilů. Ráno se systému daří dohnat noční ztrátu během přibližně dvou hodin. První ranní obměna vody spotřebuje poměrně vyšší množství vody (modrá čára klesne) a následné dopouštění vody způsobí poměrně vyšší pokles teploty (červená čára klesá s tím, jak modrá čára stoupá).

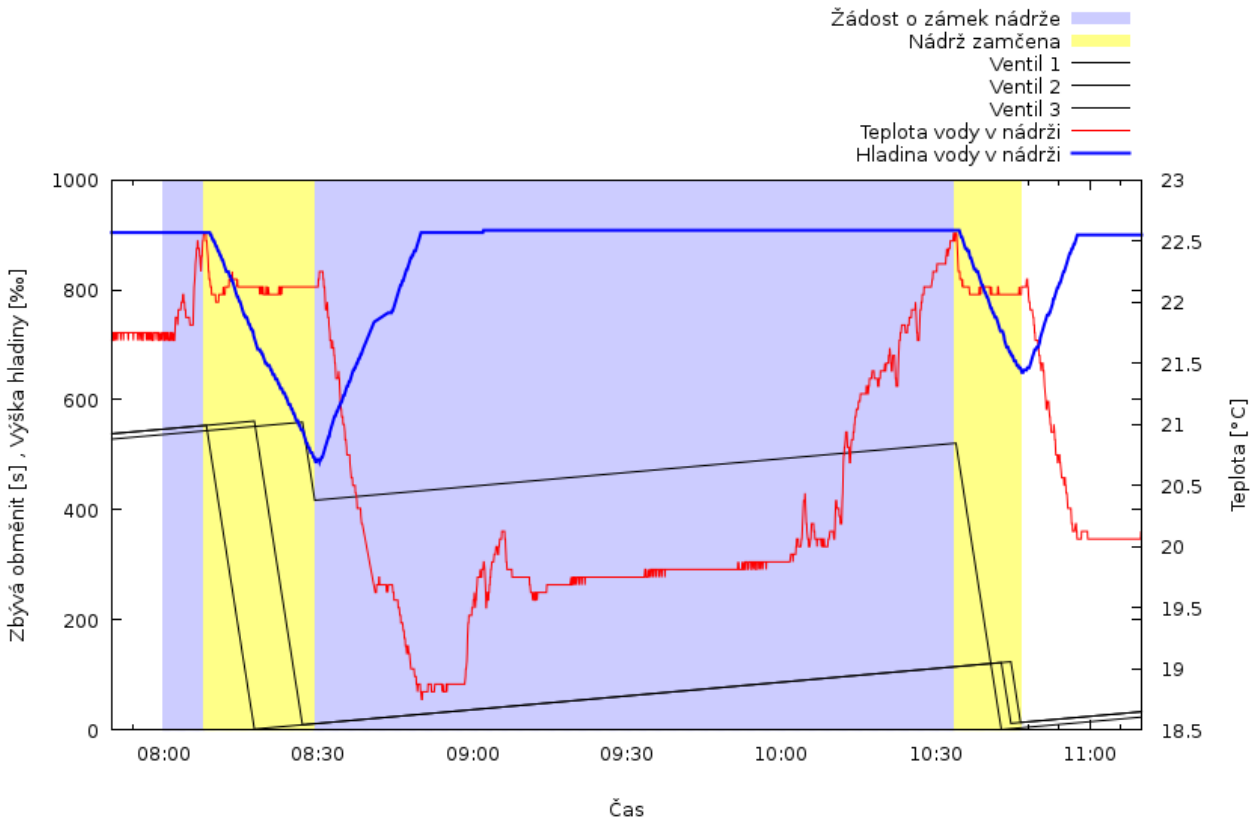
V grafu jsou vidět oblasti, kdy je vyžadováno uzamčení nádrže (modré oblasti), nádrž je napuštěna (modrá čára je na hodnotě přibližně 900 ‰), a přesto se teplota vody v nádrži (červená čára) nezvyšuje, nebo jen pomalu. Jde například o časy: 9:30, 11:50, 16:30. To je způsobeno tím, že ohřívače vody jsou napájeny pouze po čas nízkého tarifu. Po čas vysokého tarifu k ohřevu nedochází i přes to, že jednočipový počítač č.2 drží relé pro ohřev sepnuto.




**Obrázek 11.3:** *Detail obměny vody ze dne 23.4.2018*

Na obrázku 11.3 je vidět detail obměny vody v čase 23.4.2018 13:15 až 13:55.

V čase 13:23 přesáhla akumulovaná hodnota jednoho z ventilů hranici, při které se počítač č.1 začne pokoušet o obměnu vody, a tedy začne žádat počítač č.2 o uzamčení nádrže s požadovanou teplotou. Protože teplota je příliš nízká (pod 22 °C), počítač č.2 musí vodu nejdřív ohřát nad teplotu 22.5 °C (uplatňuje se hystereze 0.5 °C). To se podaří v čase 13:35 a nádrž se uzamkne. Následně začne počítač č.1 vypouštět pomocí jednoho z ventilů (toho s nejvyšší akumulovanou hodnotou). Protože během doby ohřevu vody stihla akumulovaná hodnota dalšího ventilu překročit hranici pro obměnu, počítač č.1 rovnou bez uvolnění nádrže obmění i vodu pro tento ventil. Zbývající ventil ještě nepřekročil hranici, a proto je následně již nádrž uvolněna (v čase 13:38). Protože ale hladina vody v nádrži neklesla pod hranici napouštění, počítač č.2 nezačne nádrž dopouštět. V čase 13:40 dojde k překročení hranice obměny vody i u posledního ventilu, a proto si počítač č.1 opět vyžádá uzamčení nádrže a obmění i poslední ventil. Obměna posledního ventilu již způsobí pokles hladiny v nádrži pod hranici napouštění, a proto počítač č.2 ihned po uvolnění nádrže vodu opět dopustí, což způsobí i pokles teploty. Stále však teplota neklesne pod hranici 22 °C, a proto příští uzamčení nádrže proběhne bez ohřevu vody (lze vidět na obrázku 11.2).




**Obrázek 11.4:** *Detail ranní obměny vody ze dne 24.4.2018*

Na obrázku 11.4 je detail ranní obměny vody ze dne 24.4.2018.

Z grafu je poznat, že počítač č.1 nestihne obměnit vodu pro všechny ventily na jedno uzamčení nádrže. První obměna vody spotřebuje poměrně velké množství vody, které je potřeba opět dopustit, což způsobí poměrně velký pokles teploty. Ohřátí vody po první obměně vody trvá přibližně hodinu a půl. Na druhý pokus už se podaří obměnit všechnu potřebnou vodu.

# Kapitola 12

# Závěr

Podařilo se mi navrhnout, implementovat a do provozu úspěšně nasadit vestavěný systém pro automatickou bezobslužnou obměnu vody v akváriích. Je pravděpodobné, že systém bude vyžadovat v budoucnu další vývoj podle toho, jak se budou měnit požadavky na jeho funkci, nicméně v momentálním stavu systém splňuje požadavky a funguje dle očekávání.

V budoucnu například plánujeme do systému zapojit další regály s vlastními ventily, nebo také řízení ventilace na základě měřených teplot uvnitř a venku.

Díky systému ušetříme přibližně 20 hodin pracovního času obsluhy na výměně vody za týden. Dále díky využití vrtu jako dalšího zdroje vody se podařilo snížit náklady za vodu, která byla před nasazením systému při manuální výměně vody odebírána výhradně z veřejného vodovodu.

Díky tomu, že systém automatizuje některé úkony (řízení osvětlení a větrání), se snížilo množství činností, které bylo nutné provádět manuálně. Díky možnosti vzdáleného ovládání se taktéž omezuje nutnost vstupu do odchovny pro provedení některých úkonů (zejména například neplánované změny v požadavcích na osvětlení).

# Literatura


[1] Atmel Corporation, 2011. ATmega32/L Datasheet [online]. [cit. 2017-12-18]. Dostupné z: http://www.atmel.com/images/doc2503.pdf

[2] ELEC Freaks, 2011. Ultrasonic Ranging Module HC-SR04 [online]. [cit. 2017-09-04]. Dostupné z: https://cdn.sparkfun.com/datasheets/Sensors/Proximity/HCSR04.pdf

[3] IEEE Standards Association, 2012. IEEE Standard for Ethernet [online]. [cit. 2017-12-06]. Dostupné z: https://standards.ieee.org/getieee802/download/802.3-2012_section1.pdf

[4] Jaroslav Hoffman a Jindřich Novák: *Akvaristika: Jak chovat tropické ryby jinak a lépe*. Praha: X-Egem, Nova, s.r.o., 1996, ISBN 80-7199-009-4; 80-7176-408-6.

[5] Maxim Integrated, 2015. DS18B20 Programmable Resolution 1-Wire Digital Thermometer [online]. [cit. 2017-12-18]. Dostupné z: https://datasheets.maximintegrated.com/en/ds/DS18B20.pdf

[6] Microchip, 2008. ENC28J60 Data Sheet [online]. [cit. 2017-09-04]. Dostupné z: http://ww1.microchip.com/downloads/en/DeviceDoc/39662c.pdf

[7] Microchip, 2010. ENC28J60 Silicon Errata and Data Sheet Clarification [online]. [cit. 2017-10-15]. Dostupné z: http://ww1.microchip.com/downloads/en/DeviceDoc/80349c.pdf

[8] Miroslav Kraut: *Tlamovci v akváriu*. Praha: Grada Publishing, a.s., 2008, ISBN 978-80-247-2704-2.

[9] NXP, 2007. LM75A [online]. [cit. 2017-09-04]. Dostupné z: https://www.nxp.com/docs/en/data-sheet/LM75A.pdf

[10] Smartec, 2016. Datasheet SMT172 [online]. [cit. 2017-12-18]. Dostupné z: https://smartec-sensors.com/cms/media/Datasheets/Temperature/SMT172-Datasheet.pdf

[11] WIZnet, 2011. W5100 Datasheet [online]. [cit. 2017-09-04]. Dostupné z: http://www.wiznet.io/wp-content/uploads/wiznethome/Chip/W5100/Document/W5100_Datasheet_v1.2.7.pdf